UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

PLATAFORMA DE RECOMENDACIÓN CON APRENDIZAJE POR REFUERZO

Marcos Redondo Almagro
Tutor: Rocío Cañamares Pérez
Ponente: Pablo Castells Azpilicueta

Junio 2019

PLATAFORMA DE RECOMENDACIÓN CON APRENDIZAJE POR REFUERZO

AUTOR: Marcos Redondo Almagro
TUTOR: Rocío Cañamares Pérez

Information Retrieval Group
Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2019

Resumen

Actualmente, con el desarrollo tecnológico que estamos viviendo, la cantidad de información con la que los sistemas informáticos trabajan es cada vez mayor, por lo que necesitamos mecanismos que sean capaces de filtrarla y ofrecérnosla. Es ahí donde entran los sistemas de recomendación, los cuales son capaces de realizar esas acciones.

Hace tiempo se ha comprobado que el mejor sistema de recomendación es el que combina varios sistemas de recomendación, es decir que combinando varios algoritmos efectivos se pueden conseguir sistemas que mejoran la efectividad que los algoritmos combinados tendrían por separado. En este trabajo nos enfocamos en desarrollar combinaciones de sistemas de recomendación basados en aprendizaje por refuerzo, una rama del aprendizaje automático que se centra en utilizar algún tipo de recompensa para que el sistema sepa si está cumpliendo con el comportamiento esperado o debe modificarlo.

En esta línea, nos centramos en particular en la lógica de los llamados bandidos multi-brazo. En concreto, consideramos el desarrollo de ensembles como bandidos, donde cada uno de sus brazos es un recomendador distinto. Así, en cada recomendación, uno de estos brazos es seleccionado y utilizado basándose en estrategias de aprendizaje por refuerzo.

Para la experimentación y prueba de los algoritmos investigados, se ha desarrollado una plataforma de recomendación y escucha de música, que permite que un recomendador interactúe con un usuario, buscando a) satisfacer sus gustos, b) aprender de ellos, y c) determinar qué algoritmo individual funciona mejor, ajustando dinámicamente la combinación de los algoritmos en el ensemble. Esta plataforma permite evaluar cualquier solución de recomendación, y se utiliza en este trabajo para probar los distintos métodos de aprendizaje investigados.

En el trabajo compararemos la efectividad de los bandidos con la de los recomendadores individuales que los componen, además de alguna opción más de referencia. Las soluciones se evaluarán por una parte de manera offline, utilizando conjuntos de datos públicos, simulando interacciones con los usuarios. Tras esta evaluación preliminar, llevamos a cabo una evaluación online con usuarios reales sobre la plataforma antes mencionada.

Palabras clave

Aprendizaje por refuerzo, sistemas de recomendación, relevancia, ensembles, bandido multi-brazo.

Abstract

Currently, with the technological development we are experiencing, the amount of information with which computer systems work is growing, so we need mechanisms that are able of filtering and offering it to us. This is where the recommendation systems, which are able of performing those actions, come into play.

It has long been found that the best recommendation effectiveness is obtained by combining several recommendation algorithms. combining well-configured ensemble of effective algorithms can generally achieve better effectiveness that the combined algorithms can have separately. In the present work we focus on developing combinations of recommender systems using reinforcement learning techniques. Reinforcement learning is a branch of machine learning that focuses on observing some type of reward to determine whether it is complying with the expected behavior or it should modify it.

In this line, we focus on the so-called multi-arm bandit framework. We consider the development of ensembles as bandits, where each of the arms is a different recommendation algorithm. In each recommendation, one of the arms is selected and used based on reinforcement learning strategies.

For experimentation and testing of the investigated algorithms, a music recommendation and listening platform has been developed, which allows a recommender system to interact with a user, seeking to a) satisfy her tastes in the short, mid and long term and b) determine which individual algorithm works better, dynamically adjusting the combination of algorithms in the ensemble. This platform allows evaluating any recommendation solution in online A/B testing mode and is used in this work to test the effectiveness of the investigated methods.

The effectiveness of bandits is compared to the individual recommenders that compose them, as well as a static ensemble building alternative. Solutions are first evaluated in offline experiments, using a publicly available datasets, simulating cyclic interactions with users. After this preliminary evaluation, we carry out an online evaluation with real users on the platform.

Keywords

Reinforcement learning, recommendation systems, relevance, ensemble, multi-arm bandit.

Agradecimientos

Me gustaría agradecer a Rocío y Pablo la oportunidad de trabajar con ellos y, sobre todo, la ayuda prestada durante la realización del trabajo.

También a todas aquellas personas que han participado en mi plataforma de recomendación registrándose y valorando canciones, ya que sin ellos este trabajo no podría haberse realizado.

Por último, hacer una mención especial a mi familia y amigos, ya que su apoyo ha sido muy importante para mí durante estos meses.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

A diario utilizamos aplicaciones como Netflix o Spotify, las cuales intentan simplificarnos el día a día ofreciéndonos contenido que nos puede gustar, siendo este acertado en muchas ocasiones. Esto, me hizo interesarme e investigar cómo son los mecanismos que estas aplicaciones siguen para hacer sus recomendaciones, lo cual me llevo a la idea final de este trabajo.

Todos los sistemas de recomendación se basan en la idea del aprendizaje automático, disciplina que busca generalizar el comportamiento de los usuarios a partir de sus patrones de conducta e interacción con el sistema.

Hay muchas formas de intentar emular ese comportamiento que se estudian en esta rama, pero llamo mi atención la utilizada en el aprendizaje por refuerzo, el cual pretende que un sistema software sea capaz de determinar en un entorno determinado, qué acciones debe escoger para maximizar alguna noción de premio o recompensa a medio o largo plazo.

Fue precisamente ese concepto de recompensa, el que despertó mi curiosidad por conocer más acerca de este tipo de aprendizaje. Podría compararse con la forma de aprender de un animal doméstico al que si tiene un comportamiento positivo se le premia con una "recompensa", lo que favorece que siga manteniendo esa conducta, mientras que, si este es negativo, la ausencia de recompensa le induce a aprender que ese no es el comportamiento que tienen que seguir.

1.2 Objetivos

El objetivo del presente trabajo de fin de grado consiste en crear una plataforma de recomendación que compare, de manera online, el comportamiento de algunos recomendadores que utilizan técnicas de aprendizaje por refuerzo, frente a otros recomendadores de referencia. Siendo más concretos los objetivos principales serán:

- Recopilar información teórica sobre sistemas de recomendación y aprendizaje por refuerzo para poder decidir qué recomendadores podrían ser utilizados en la plataforma.
- Buscar la configuración óptima para cada uno de los recomendadores a implementar de manera offline, realizando barridos sobre los parámetros de configuración de cada uno de ellos, para así intentar optimizar los resultados online.
- Implementar la plataforma de recomendación interactiva adaptándola al conjunto de datos, para así poder ofrecer recomendaciones y obtener valoraciones de sus ítems de forma online.
- Cuantificar y analizar el acierto de los usuarios en sus valoraciones para cada uno de los recomendadores y comparar dichos resultados, tanto de manera offline como online.
- Comprobar si los recomendadores que aplican aprendizaje por refuerzo mejoran a los recomendadores individuales que se suelen utilizar, así como a otros que también combinen varios recomendadores.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- En el capítulo 2, exponemos el estado del arte de los sistemas de recomendación, así como del aprendizaje por refuerzo, para poder contextualizar alguno de los conceptos que se abordan a lo largo de este trabajo.
- En el capítulo 3, presentamos las decisiones tomadas en la implementación de cada uno de los recomendadores, así como un análisis de los resultados obtenidos simulando de manera offline para tener una aproximación de lo que se espera con los resultados de la plataforma.
- En el capítulo 4, mostramos detalles sobre la implementación y el funcionamiento de la aplicación y analizamos los resultados de los datos obtenidos de los usuarios reales.
- En el capítulo 5, terminamos exponiendo de manera sintetizada las conclusiones obtenidas de los resultados de las pruebas y exponemos las bases de lo que podría ser un trabajo futuro.

2 Estado del arte

2.1 Introducción

Actualmente, con los avances tecnológicos que surgen día a día, no nos imaginamos una vida sin nuestro ordenador, móvil o reloj inteligente y, por consiguiente, sin muchas de sus aplicaciones. Esto ha provocado que la cantidad de información a la que tenemos acceso se haya incrementado desmesuradamente, siendo imposible para una persona encontrar lo que busca en un tiempo razonable. Por ese motivo, surgen los sistemas de recomendación, ya que es necesario algún tipo de mecanismo que procese dicha información, la filtre y se la ofrezca al usuario de una forma rápida. Por ejemplo, en el campo del ocio, surgieron aplicaciones que utilizan dichos sistemas, como **Netflix** o **Spotify**, en el campo comercial de la mano de **Amazon**.

Estos sistemas, necesitan información del usuario para poder personalizar su búsqueda para la recomendación, pero inicialmente no se conoce nada sobre él ni de sus gustos o necesidades. Para solventar este problema, utilizan técnicas de **aprendizaje activo o por refuerzo**, las cuales permiten ofrecer recomendaciones, a la vez que obtienen información del usuario para optimizar las siguientes.

2.2 Sistemas de recomendación

Los sistemas de recomendación se definieron originalmente como aquellos en los que "las personas proporcionan recomendaciones como entrada, que luego el sistema añade y dirige a los destinatarios apropiados" (Resnick et Varian, 2005). El término ahora tiene una más amplia connotación, describiendo cualquier sistema que produzca recomendaciones individualizadas como salida o que guie al usuario de forma personalizada, hacia ítems útiles e interesantes en un amplio espacio de opciones (Burke, 2005).

Estos sistemas gestionan conjuntos de **usuarios**, a los cuales se les quiere recomendar **ítems**, en base los **ratings**, que son las puntuaciones que los usuarios otorgan a dichos ítems.

Cuando la puntuación refleja una preferencia favorable al ítem decimos que ese ítem es **relevante** para el usuario, siendo **no relevante** en el caso contrario. Los ratings pueden ser tanto binarios (me gusta / no me gusta) como numéricos (una escala de números de menor a mayor gusto por el ítem). En el caso de los binarios es claro como distinguir cuales son relevantes (sí o no), pero para los numéricos es necesario utilizar una regla para delimitar cuales se consideran relevantes y cuáles no (típicamente poner uno de sus valores numéricos como umbral de relevancia).

Existen dos formas de recoger estas valoraciones. Una es de forma implícita, extrayendo la información pertinente de las acciones del usuario. Por ejemplo, el tiempo que pasa leyendo una determinada página web, los enlaces que sigue, el número de veces que se escucha una canción. La segunda, utilizada en este trabajo, es explícitamente, es decir el usuario asigna una puntuación a cada elemento que sería un valor numérico discreto entre un máximo y un mínimo (Nichols, 1998).

2.2.1 La tarea de recomendación

Son varios los pasos que sigue el sistema para poder realizar una recomendación, las cuales pueden dividirse en:

- **Observar** al usuario mientras éste realiza actividades.
- **Detectar** patrones de comportamiento, identificando indicios de interés del usuario.
- **Predecir** posibles intereses futuros basados en indicios de intereses pasados.
- **Sugerir** al usuario las opciones que cree que pueden ser más de su interés.

Para poder llevarlas a cabo los sistemas de recomendación necesitan tener alguna forma de clasificar cada uno de los ítems que pueden ser recomendados, de manera que se le ofrezca al usuario el óptimo en cada momento. Para ello, haciendo uso de las anteriores puntuaciones, así como del correspondiente algoritmo de recomendación, se genera un **ranking** para ordenar los ítems que se ofrecerán al usuario, el cual, además, se irá actualizando para cada recomendación. Normalmente se considera no hacer recomendaciones repetidas, es decir, no recomendar algo que el usuario ya ha votado, aunque depende para lo que sea necesario el sistema, esta situación se puede permitir. En el caso del presente trabajo no se realizan recomendaciones repetidas.

Para almacenar las puntuaciones de los ítems que van realizando cada uno de los usuarios, es necesario una estructura que sea capaz de manejar una gran cantidad de datos de manera eficiente. Normalmente, para representarlo de forma abstracta se utiliza una **matriz de puntuaciones**. Esta consiste en una tabla en la que sus filas se corresponden con cada uno de los usuarios del sistema, sus columnas con los ítems (o viceversa) y sus entradas con las puntuaciones de ese usuario para ese ítem. En la representación de esta matriz, muchas de las entradas estarán vacías. Esto significa que el usuario aún no ha valorado ese ítem, por lo que todas esas entradas vacías para la fila (usuario), serán las posibles recomendaciones que se vayan a hacer (Jure, 2011).

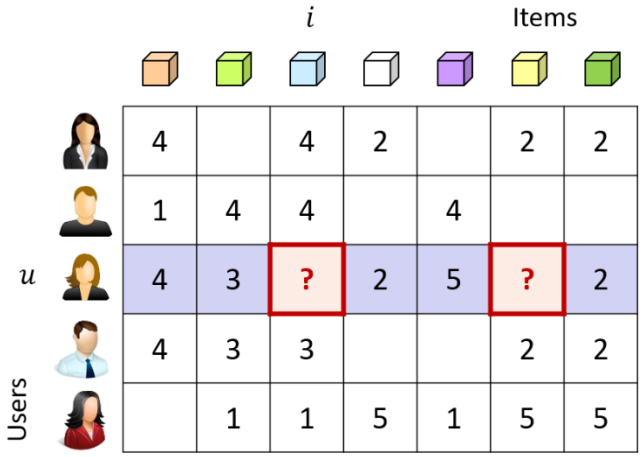

Figura 1. Ejemplo de matriz de puntuaciones

Para decidir cuál de las celdas recomendar se utilizan varios algoritmos de recomendación que serán explicados en la siguiente sección.

2.2.2 Algoritmos de recomendación

Existen numerosos algoritmos de recomendación los cuales parten de la misma premisa y objetivo, pero que difieren entre ellos en la forma de elegir y ordenar los ítems que se van a recomendar.

Todos ellos parten del mismo funcionamiento:

- Se le pasan como entrada al algoritmo las puntuaciones que ya existen en el sistema, el usuario y el número máximo de ítems que se quieren recomendar (tamaño del ranking).
- El algoritmo realiza las acciones pertinentes y asigna a cada ítem conocido por el sistema una puntuación, la cual también conocemos como **score**. Es precisamente en los cálculos de esos scores en lo que se diferencian los algoritmos, ya que cada uno busca maximizar cosas diferentes.
- Se ordenan de mayor a menor dichos scores y se devuelve en un ranking con el tamaño (número de ítems a recomendar) que se había pasado como entrada.

Dependiendo del contexto del sistema en el que se ejecuta la recomendación, podemos distinguir dos clases de algoritmos (Aggarwal, 2016):

- **Basados en contenido**: Además de la interacción observada (o ratings) entre usuarios e ítems, se suelen emplear las características de estos ítems, como entrada a estos algoritmos. Básicamente consiste en que, si dos objetos tienen características parecidas y al usuario le ha gustado uno de esos objetos, es probable que le guste también el otro debido a la similitud entre sus características.
- **Basados en filtrado colaborativo**: Utilizan funciones de similitud entre todos los usuarios o ítems para realizar recomendaciones. Estas se calculan en función de las puntuaciones que los usuarios asignan y no de las características de los ítems como los anteriores. En este caso, al usuario se le recomiendan ítems que le gustan a usuarios parecidos a él o parecidos a los que ya le han gustado.

En el presente trabajo, únicamente se utilizan algoritmos basados en filtrado colaborativo, por lo que los algoritmos que serán presentados posteriormente pertenecen todos a este grupo. Los algoritmos de filtrado más utilizados son:

- **Vecinos próximos** (**kNN**): Este algoritmo puede ser tanto orientado a ítems como a usuarios. Básicamente consiste en crear un vecindario con los N vecinos más próximos (más similares) al ítem o usuario con el que se compara. Estos, por tanto, serán usados en orden de mayor a menor similitud para construir el ranking para la recomendación.
- **Factorización de matrices:** Este algoritmo intenta, partiendo de la matriz de puntaciones del sistema, descubrir características latentes (ocultas en el sistema), conocidas como factores, que concuerden con las puntuaciones obtenidas. Cada componente del vector indica el peso de ese factor en un usuario o ítem y es por medio de ese peso, por el que se obtiene la puntuación para construir el ranking.

La noción de sistema de recomendación suele llevar implícita la suposición de que es un sistema personalizado, es decir que genera una salida diferente adaptada para cada usuario, considerando que los gustos individuales son diferenciados. Sin embargo, es no tiene por qué ser siempre así, y puede ser útil considerar recomendadores que no tienen en cuenta la

individualidad de los usuarios a la hora de decidir a quién enviar una lista de ítems recomendados.

Una recomendación no personalizada común es por orden (decreciente) de **popularidad**, donde la popularidad de un ítem se puede definir como el número de usuarios a quienes les gusta el ítem, o su puntuación promedio. En ocasiones puede ser una opción razonable e incluso efectiva en sistemas reales, por ejemplo, cuando la densidad de datos es extremadamente baja, o para generar recomendaciones para usuarios nuevos. Otra recomendación útil, sólo como referencia experimental, que utilizaremos en este trabajo, es la **recomendación aleatoria**: permite, por ejemplo, detectar que un algoritmo no funciona cuando su rendimiento se distingue poco del aleatorio.

2.2.3 Arranque en frío

Puede darse la situación en la que un algoritmo de recomendación no tenga suficiente información sobre un usuario o ítem, lo cual produce que sea complicado poder realizar una recomendación válida. Un ítem sin ratings no puede recomendarse por filtrado colaborativo, pero sí mediante métodos basados en contenido (si disponemos de información sobre las características del ítem). Por otra parte, a un usuario sin ratings sólo pueden hacérsele recomendaciones no personalizadas, como la recomendación de ítems populares.

Esta es una de las causas que motiva el uso de soluciones híbridas. Puede, por ejemplo, combinarse un algoritmo de filtrado colaborativo con uno basado en contenido, para que este último pueda detectar similitudes entre ítems, sin necesidad de que los usuarios hayan realizado valoraciones anteriores (Schein et al, 2002). Por otra parte, el filtrado colaborativo y/o basado en contenido se puede combinar con recomendaciones populares para poder atender igualmente a usuarios nuevos o antiguos. El presente trabajo no se centra en utilizar este tipo de algoritmos para solucionar el problema de arranque en frío, sino para mejorar la recomendación, pero es otra de las grandes utilidades que tiene.

2.2.4 Algoritmos híbridos

Estos algoritmos surgen con el objetivo de cubrir las debilidades que pueden tener cada uno de los algoritmos de recomendación por separado, para potenciar y mejorar la recomendación que hace el sistema. Por tanto, podemos catalogar como un algoritmo de este tipo a cualquiera que combine múltiples técnicas de recomendación para producir una única salida.

Es común, por ejemplo, como acabamos de mencionar, la combinación entre algoritmos basados en contenido y filtrado colaborativo, aunque puede hacerse con cualquier tipo de algoritmo de recomendación. Entre las diferentes formas de combinar algoritmos destacan (Burke, 2007):

- **Ponderación**: La puntuación de los diferentes componentes de la recomendación se combinan numéricamente: se suman con un peso propio para cada algoritmo.
- **Conmutación**: Se elige entre los componentes de la recomendación y se aplica el seleccionado.
- **Mixto**: Las recomendaciones de los diferentes algoritmos se presentan juntas.
- **Combinación de características**: Las características derivadas de las diferentes fuentes de conocimiento se combinan y se asignan a un único algoritmo de recomendación.

- **Aumento de características**: Se utiliza un algoritmo de recomendación para calcular una característica o un conjunto de características, que luego formará parte de la entrada al siguiente.
- **Cascada**: Se le da una prioridad estricta a los recomendadores, teniendo una prioridad más baja aquellos que rompen empates en la puntuación de los superiores.
- **Meta-nivel**: Se aplica un algoritmo de recomendación y se produce algún tipo de modelo, que será la entrada utilizada por el siguiente algoritmo.

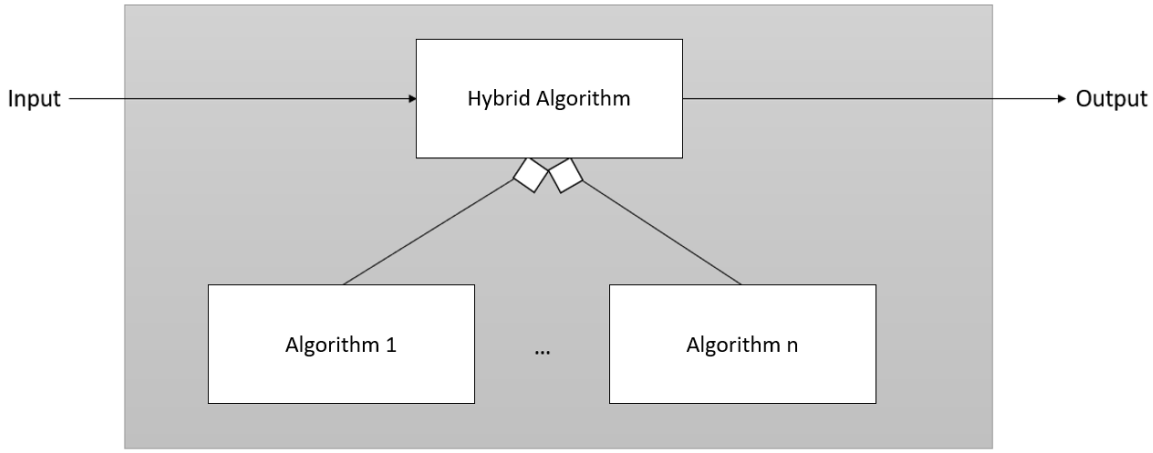

Figura 2: Lógica de un algoritmo de recomendación híbrido

En este trabajo nos centraremos en la hibridación por conmutación, dado que se adapta de forma natural al paradigma de los bandidos multi-brazo, si bien la compararemos con soluciones de ponderación.

2.2.5 Ensemble en algoritmos de recomendación

Los ensembles de recomendadores son un caso particular de los híbridos en los que los algoritmos combinados se consideran como una caja negra. Esto permite a diferencia de otros algoritmos híbridos, que no se realice ningún tipo de preprocesamiento de todos los algoritmos que formen uno único, sino que directamente se usen las salidas de cada uno de ellos. Esto tiene la gran ventaja de permitir la combinación de tantos algoritmos como queramos y que estos sean de cualquier tipo.

La forma más utilizada por este tipo de algoritmos consiste en combinar las salidas (scores) obtenidas de cada algoritmo ejecutado individualmente, ya sea por suma de puntuaciones o por posición en el ranking de cada ítem, y realizar un nuevo ranking con esos resultados ya combinados. Alternativamente, otra forma podría ser usar en cada momento solo la recomendación devuelta por uno de ellos, eligiendo el "mejor" basándonos en alguna métrica de "calidad" en la recomendación (Adomavicius et Tuzhilin, 2005).

Cuando se suman las puntuaciones es muy conveniente normalizarlas previamente, ya que cada algoritmo puede utilizar escalas y distribuciones muy diferentes en sus puntuaciones. Un método de normalización muy común y efectivo es **Rank-sim** (Joon, 1997), que normaliza las puntuaciones al intervalo [0,1] teniendo en cuenta sólo las posiciones de los ítems en la recomendación, ignorando la puntuación numérica del sistema que define el ranking de recomendación. Esta operación se realiza mediante la siguiente fórmula:

$$\text{Rank} - \text{sim}(\text{ítem}) = \frac{posicion\ de\ item\ en\ el\ ranking - 1}{tamaño\ del\ ranking}$$

En caso de que un ítem no se encuentre en alguno de los rankings, se contabiliza como 0. Una vez realizados los cálculos para cada ítem, se suman las puntuaciones normalizadas de cada ítem en cada uno de los rankings, dando lugar por esta suma a un único ranking que será el utilizado para la recomendación. Lo esperable por tanto es que los ítems que aparecen en los rankings de los algoritmos combinados y se encuentran en las primeras posiciones en cada uno de ellos queden en las posiciones más altas (Joon, 1997).

2.2.6 Evaluación

La evaluación de los sistemas de recomendación consiste en medir y comparar la calidad que tienen las recomendaciones generadas por diferentes algoritmos. Existen muchas formas de medir dicha calidad dependiendo de lo que se busque optimizar.

Una de estas formas de evaluar son las **métricas de error**. Estos comparan las puntuaciones que el recomendador predice, con la puntuación real que el usuario ha dado a un determinado ítem. Entre ellas destacan:

$$\text{Root Mean Squared Error (RMSE)} = \sqrt{\frac{\sum_{i=0}^{N-1}(\hat{u}_i - u_i)^2}{N}}$$

$$\text{Mean Absolute Error (MAE)} = \frac{\sum_{i=0}^{N-1}|\hat{u}_i - u_i|}{N}$$

También se vienen utilizando otras formas como son las **métricas de ranking,** métricas que se centran en la calidad de dicho ranking. Dentro de estas se pueden distinguir tres tipos principales, dependiendo de lo que se pretenda optimizar:

- **Precisión**: Mide el grado de acierto de las recomendaciones realizadas a los usuarios.
- **Diversidad**: Evalúa en qué medida la recomendación cubre o no una amplia variedad de ítems, atendiendo a las características de éstos.
- **Novedad**: Mide la capacidad de recomendar ítems que no resulten excesivamente obvios, predecibles, familiares o conocidos para el usuario.

En este trabajo nos enfocamos al acierto en la recomendación, así que nos centraremos en las métricas de precisión. Aquí, entra en juego la noción de **relevancia**, mencionada anteriormente, ya que estas predicciones son binarias. Por tanto, si un ítem recomendado es relevante, se le considera como un acierto. Entre estas métricas destacan:

$$\text{Precision} = \frac{|Recomendados\ \cap\ Relevantes|}{Recomendados}$$

$$\text{Recall} = \frac{|Recomendados\ \cap\ Relevantes|}{Relevantes}$$

Estas se pueden también delimitar a un tamaño concreto de ranking K, siendo representadas como P@K y R@K y evaluando únicamente, los objetos del ranking que se encuentren en ese top (Valcarce et al, 2018).

Para evaluar un sistema de recomendación, ya sea mediante métricas de error o de ranking, hay que distinguir si este esta va a ser **offline** u **online**. En el primer caso, se dispone de un conjunto de datos, que comúnmente se dividen en un conjunto de **entrenamiento**, que utilizan como entrada los algoritmos de recomendación, y un conjunto de **test**, que se oculta a los algoritmos evaluados, y permite comparar estos datos con la recomendación devuelta por el algoritmo para así comprobar si es relevante para el usuario. En el segundo caso, se utilizan todos los datos que tiene el sistema hasta el momento para generar recomendaciones, y es la puntuación que da el usuario en respuesta a la recomendación la que se utiliza directamente para evaluar la relevancia del resultado (Cañamares et Castells, 2018).

2.3 Aprendizaje por refuerzo

Este tipo de aprendizaje es una rama del **aprendizaje automático**, el cual tiene como objetivo desarrollar técnicas para que los sistemas inteligentes puedan aprender de forma autónoma, es decir, que sean capaces de generalizar comportamientos a partir de información que le sea suministrada. El **aprendizaje por refuerzo** explora la forma de asignar esos comportamientos a acciones, las cuales buscarán maximizar una recompensa de forma numérica. De esta forma, al sistema no se le dice qué acciones debe realizar, sino se le intenta mostrar qué acciones producen mayor recompensa para que él decida qué hacer.

Estas acciones pueden afectar no solo a la recompensa más inmediata sino también a las sucesivas, por lo que estas recompensas van actualizándose de forma dinámica. Estas dos características, tanto la búsqueda por prueba y error, como la recompensa diferida, son los dos rasgos distintivos del aprendizaje por refuerzo respecto a otros tipos de aprendizaje automático (Sutton et Barto, 1998).

2.3.1 Formulación de la tarea

Antes de entrar en la lógica que sigue este tipo de aprendizaje, hay que mencionar los elementos por los que está formado, entre los que destacan:

- Un agente (algoritmo)
- Un entorno (sistema)
- Un instante de tiempo (momento en que se produce la valoración)
- Un conjunto de estados de entorno (posibles elecciones de ítem)
- Un conjunto de acciones (elección de uno de los ítems posibles)
- Una recompensa (valor numérico de lo óptimo que ha sido realizar la acción)

Su funcionamiento consiste en la aplicación de un **Proceso de Decisión de Markov (MDP)** (Sutton et Barto, 1998), el cual se basa en los siguientes pasos:

- El agente recibe un conjunto finito de estados, S, y acciones, A, con los que interactuar.
- En cada instante de tiempo, T, el agente recibe un estado concreto, S_t, que normalmente incluye la recompensa R_t.
- Una vez lo ha recibido, elige una acción entre las posibles, A_t, teniendo en cuenta la recompensa junto a otros factores probabilísticos, obteniendo un nuevo estado $S_{t+1}=A_t$ (S_t).
- El entorno responde a la acción tomada por el agente A_t, por medio de una recompensa, o castigo, R_{t+1}, la cual, enviada junto con el estado al agente para trabajar en el siguiente instante de tiempo, $t+1 = (S_{t+1}, R_{t+1})$.

- Se repite el proceso mientras el agente así lo decida.

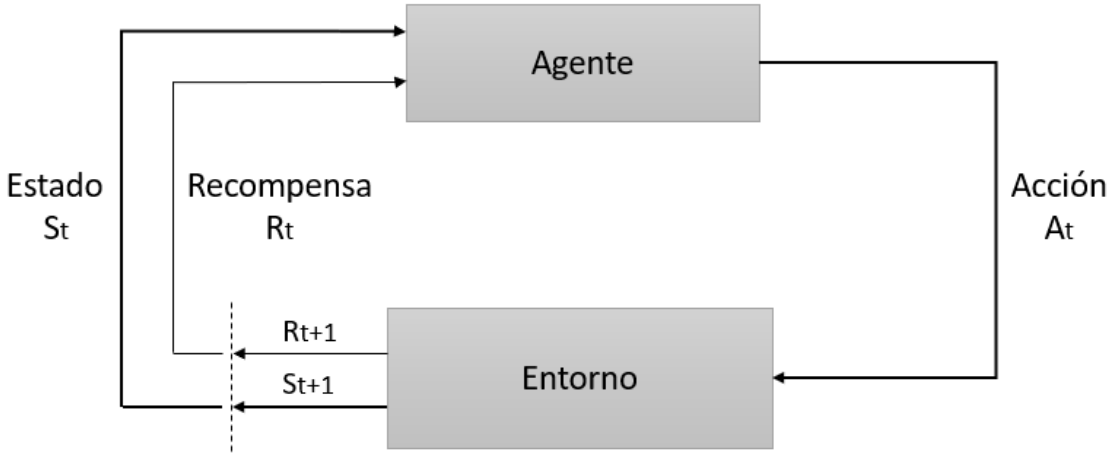

Figura 3: Interacción agente-entorno en MDP

El objetivo del agente es maximizar la recompensa, para obtener en todo momento el resultado óptimo para el entorno. Cabe destacar que el agente puede elegir cualquier acción en función de la recompensa, pero también puede aleatorizar su elección, lo cual puede resultar interesante en el proceso de explotación y exploración del que hablaremos en la siguiente sección.

2.3.2 Algoritmos de aprendizaje por refuerzo

En la descripción del algoritmo general anterior, se puede ver como el algoritmo necesita elegir entre varias acciones en cada instante de tiempo. Para tomar dicha decisión y seleccionar la opción que más le conviene, hace uso de la recompensa recibida en cada momento. Pero esta recompensa puede ser obtenida de diferentes formas, por lo que también existen diferentes algoritmos para calcularla.

Entre ellos destaca el **bandido multi-brazo**. Este consiste en un sistema con N brazos, en el que cada brazo es una de esas acciones y el objetivo de todos ellos es maximizar la recompensa a largo plazo. Se define como **bandido,** por tanto, al conjunto de brazos que conforman el sistema**.** Estos, suelen ser elementos no estacionarios, ya que las recompensas que van obteniendo cada uno de ellos, cambian dinámicamente según se van sucediendo las recomendaciones en el sistema. Este dinamismo existente en cada iteración provoca que no siempre sea elegido el mismo brazo, sino que, dependiendo de la recompensa anterior pueda haber variaciones en esa selección.

Figura 4: Figura ilustrativa de la lógica de bandidos multi-brazo

Una aplicación importante de este tipo de algoritmos son los test A/B (Siroker et Koomen, 2015), ya que al igual que hacen ellos, se decide automáticamente entre varios algoritmos en función de su rendimiento anterior. Si una elección es claramente peor que las otras, el algoritmo reducirá progresivamente el tráfico que se le asigna. Por tanto, se asume implícitamente que alguno de los algoritmos será llamado cada vez con menos frecuencia, ocurriendo en algún caso, que no haya un ganador claro o que el mejor algoritmo en ese momento no lo sea en el futuro.

En el caso de los test A/B, ese algoritmo que va perdiendo frecuencia, acaba siendo 0, pero los bandidos, son capaces de controlar esas situaciones pudiendo revertir la situación si el algoritmo así lo cree necesario. Además, los test A/B, suelen ser muy costosos computacionalmente cuando se tiene que elegir entre un número elevado de brazos ya que tienen que ejecutarse todos ellos para decidir cuál elegir, algo que solventan los bandidos, ya que estos, únicamente seleccionan y ejecutan el brazo elegido en base a la acción anterior.

Por eso, para este tipo de algoritmo y, por tanto, para el aprendizaje por refuerzo en general, cobra mucha importancia los términos **explotar** y **explorar**, ya que permite descubrir nuevas posibilidades que puedan maximizar la recompensa, a la vez que se van eligiendo las óptimas. El algoritmo explota si escoge el brazo con la recompensa más alta estimada, basada en la recompensa anterior. La exploración es contraria, ya que, en este caso, el algoritmo escoge un brazo que no tiene la mayor recompensa, pero que puede descubrir una recompensa mejor en futuras iteraciones. Una de las principales preocupaciones para este algoritmo es realizar un buen balanceo entre estas dos posibilidades, para obtener mejores resultados con el paso de las recomendaciones (Liang, 2017).

Para tratar este problema, además del propio de obtener la mejor recompensa existen varios algoritmos para elegir el brazo adecuado entre los que destacan:

- **ϵ-greedy** (Koulouriotis et Xanthopoulos, 2008): Es el algoritmo más simple. Utiliza un único parámetro ϵ, el cual puede variar entre $0 < \epsilon < 1$, y en cuya configuración estándar es fijo. Con probabilidad 1 - ϵ, se escoge el brazo que es mejor en ese momento (explotación), mientras que con probabilidad ϵ, se escoge un brazo aleatorio (exploración). Esa probabilidad se suele escoger de manera aleatoria para cada iteración. El algoritmo puede tener varias variaciones, como por ejemplo ir variando ϵ, en función de lo que se va aprendiendo.
- **Thompson sampling** (Brodén et al, 2018): Este algoritmo modela una distribución de probabilidad para obtener la recompensa de cada uno de los brazos, basándose en las observaciones previas. Si el valor de la recompensa es binario, se utiliza una distribución beta-bernoulli. Esta cuenta con dos parámetros para cada uno de los brazos, α y β, que irán aumentando en caso de acierto o fallo respectivamente en el brazo correspondiente. Estos controlan la forma que seguirá la distribución en cada momento, siendo por ejemplo la media, $\mu = \alpha/\alpha + \beta$. Así, partiendo de estos parámetros, en cada recomendación, se extrae una muestra (normalmente dicha media), de la distribución de cada brazo y se elige el brazo con el mayor valor muestral.
- **Softmax** (Liang, 2017): Una desventaja del algoritmo ϵ-greedy es que no distingue entre los brazos con peor resultado. Este algoritmo, sí que tiene en cuenta esta diferencia. La probabilidad de elegir un brazo es igual a $e^{r/t} / \sum_{i=1}^{n} e^{r_i/\tau}$ donde r es la recompensa estimada para el brazo, n es el número de jugadas, y τ es un parámetro que varía entre 0 a 1 y que cumple una función similar al ϵ de ϵ-greedy.

- **UCB** (Liang, 2017): A diferencia de los algoritmos anteriores, a UCB no solo le preocupa la recompensa de las jugadas anteriores, sino también lo que sabe acerca de las armas disponibles. En la fase inicial, UCB explora cada brazo al menos una vez. Después de eso, cada vez que toca el brazo con el valor más alto de la fórmula $r + b$, donde r es la recompensa estimada para ese brazo, y b es una bonificación especial (límite de confianza) para ese brazo. En UCB, los brazos que se exploran menos en las jugadas anteriores recibirán una mayor bonificación, mientras que los que se exploran más recibirán una menor.

En algunos sistemas, tenemos algún tipo de contexto durante la elección, como por ejemplo la edad o el género, lo cual nos puede ayudar a identificar mejor el brazo a escoger. Para estos casos también existe un algoritmo que aborda el problema, el cual es conocido como bandido contextual multi-brazo.

En este caso, para cada iteración, se analiza un vector de características (vector de contexto) junto a la recompensa para hacer la elección del brazo en esa iteración. El objetivo por tanto es ir recopilando suficiente información sobre cómo se relacionan entre sí los vectores de contexto y las recompensas, para poder predecir cuál es el mejor brazo (Langford et Zhang, 2008).

2.3.3 Bandidos multi-brazo en recomendación

Los bandidos multi-brazo surgen para elegir entre varias opciones dinámicamente y por tanto pueden ser aplicados claramente al diseño de sistemas de recomendación, donde también existe una necesidad de elegir entre varias opciones: los ítems a recomendar. De hecho, la aplicación de técnicas de bandidos multi-brazo ha comenzado a popularizarse cada vez más en este campo. El trabajo en esta área se ha centrado en utilizar como brazos los ítems a recomendar, buscando solucionar el problema de la selección del ítem más adecuado para el usuario en cada momento en que se realiza la recomendación (Li et al, 2016).

Por tanto, el objetivo del bandido es decidir sobre una secuencia de acciones (en este caso ítems), aquella que maximice el valor de recompensa obtenido. En este caso la recompensa es modificada según si se acierta o no en la recomendación del ítem seleccionado, de forma que para futuras recomendaciones se tiene en cuenta el éxito o fracaso previo de recomendar cada ítem.

En el trabajo, proponemos una forma alternativa de aplicar este tipo de bandidos en recomendación y en vez de utilizar ítems, utilizaremos algoritmos de recomendación como brazos, con lo cual el bandido multi-brazo resultante hace las funciones de ensemble.

3 Ensembles multi-brazo de recomendación

Como ya avanzamos al final del capítulo 2, este trabajo proponemos una forma alternativa de trabajar con los bandidos multi-brazo y es utilizar recomendadores como brazos en vez de ítems. Esta reformulación nos permite transformar cualquier bandido multi-brazo en un ensemble dinámico de recomendadores, mejorando la efectividad de cualquiera de los recomendadores que combina por separado, ya que se beneficia de su capacidad para adaptarse dinámicamente al candidato que mejor esté funcionando.

En cada recomendación, el bandido elige un recomendador basado en un modelo de su efectividad y obtiene de dicho recomendador el ítem a recomendar. Este modelo se actualiza interactivamente con la respuesta de los usuarios a la recomendación que es emitida por el recomendador seleccionado, permitiendo que otro brazo pueda ser elegido en la siguiente recomendación.

A lo largo de este capítulo definiremos en detalle qué métodos han sido desarrollados para la implementación de los recomendadores que se usarán en la plataforma y cómo se realizado dicho desarrollo. También, se presentarán diversos experimentos realizados de manera offline para mostrar por qué se han elegido determinados parámetros en los recomendadores utilizados en la aplicación además de verificar el comportamiento y la efectividad de dichos recomendadores, teniendo así una referencia de cara a la evaluación online que se realizara mediante la aplicación.

3.1 Recomendadores de aprendizaje por refuerzo

La implementación de estos recomendadores está basada en un **bandido multi-brazo**, donde cada uno de los brazos que lo componen, es un recomendador de referencia (recomendador individual básico).

El comportamiento general de estos recomendadores de aprendizaje por refuerzo consiste en que, para cada iteración, se escoge uno de los brazos en base a la recompensa de la iteración anterior (la cual varía según si ha sido acertada o no la recomendación), y una vez seleccionado, se utiliza para la recomendación, el ranking que el recomendador de ese brazo ha devuelto. Es importante dejar claro que el ranking de cada recomendador es independiente, eligiéndose dinámicamente en cada iteración según el algoritmo del bandido implementado.

Para el desarrollo del trabajo utilizamos dos recomendadores de este tipo, pero con diferentes estrategias: ϵ-greedy y Thompson sampling. Se quiere comprobar, por tanto, el rendimiento de estos recomendadores entre ellos y, además, compararlos con otros más básicos que no realizan ningún tipo de aprendizaje para ver la influencia de la recompensa en la recomendación. A continuación, se explicará el funcionamiento de estos dos recomendadores.

3.1.1 Recomendador ϵ-greedy

Este recomendador sigue la lógica del algoritmo ***ϵ*-greedy**, utilizado por los bandidos multi-brazo. En este caso los brazos que hemos utilizado son los recomendadores de factorización de matrices y voto promedio, que presuntamente, son dos de los que deberían ofrecer mejores resultados.

Para elegir el brazo a utilizar se escoge una probabilidad entre 0 y 1 de manera aleatoria. Si esa probabilidad es mayor que ϵ, se escoge el brazo que es considerado mejor en ese

momento. Para saber cuál de ellos es, se va contabilizando el número de aciertos y fallos en las valoraciones obtenidos hasta ese momento y de estos valores, se obtiene una media (acierto / (acierto + fallo)) con la que se decide el brazo elegido. Por el contrario, si queda por debajo de ϵ, se vuelve a escoger una probabilidad entre 0 y 1 de manera aleatoria, siendo escogido uno u otro recomendador con la misma probabilidad (50% - 50%).

De esta forma, como normalmente se usa una ϵ baja, la mayoría de las veces se realizará explotación, usando el mejor recomendador, pero también permitirá que en ocasiones se realice exploración, pudiendo descubrir otros ítems del gusto del usuario, que el otro recomendador puede que no contemplase y que a la larga maximicen la recompensa (Elahi et al, 2016).

3.1.2 Recomendador Thompson sampling

Este recomendador sigue la lógica de otro de los algoritmos de los bandidos multi-brazo como es **Thompson sampling**. Al igual que en el anterior, los brazos que hemos utilizado son los recomendadores de factorización de matrices y voto promedio.

Al ser el recomendador binario, es decir, a la hora de evaluar la valoración la cataloga como relevante o no, utilizamos una muestra de una distribución beta-bernoulli para aproximar la recompensa. En este caso, esa muestra escogida es la media, la cual es calculada utilizando los valores de α y β en cada una de las iteraciones. Con este valor, se elige que brazo utilizar para la recomendación, actualizándose los valores de α y β cada vez que el usuario realiza una valoración.

3.2 Recomendadores de referencia

Para la implementación de los recomendadores basados en aprendizaje por refuerzo, hemos utilizado varios ya existentes para sus brazos. Los recomendadores como ya hemos comentado son **aleatorio**, **voto promedio y factorización de matrices**. Estos son utilizados también como recomendadores independientes en los resultados finales, para así comprobar si los recomendadores combinados implementados, los mejoran. A continuación, explicaremos como han sido implementados además de cómo funcionan.

3.2.1 Recomendador aleatorio

Este es el más simple de todos los recomendadores. Básicamente a cada uno de los ítems disponibles para recomendar le asigna un score aleatorio entre 0 y 1 y los coloca en el ranking de recomendación ordenándolos de acuerdo con dicho score. Sin ser combinado, nos sirve como un recomendador informativo, ya que permite saber cuál es el priori de relevancia, es decir, que resultado se obtendría sin tener en cuenta las opiniones disponibles de los usuarios, esto es, sin emplearlas para producir ninguna señal que pueda resultar útil de cara a la recomendación. Se puede decir, por tanto, que hace las funciones de 0 del sistema de referencia.

3.2.2 Recomendador voto promedio

Se basa en recomendar lo más popular para la mayoría de los usuarios que están en el sistema. Existen muchas variaciones de algoritmos basados en popularidad, pero en concreto hemos utilizado uno que se enfoca en la puntuación media de los ítems por parte de los usuarios, conocido como voto promedio. Existen otras versiones populares como por ejemplo las que recomiendan basándose en el número de usuarios que han valorado el ítem, pero en este caso, en el conjunto de datos utilizado todas las canciones tienen el mismo número de votos, por lo que no es una señal discriminativa.

Otra de las opciones que se adaptan a este conjunto habría podido ser calcular la popularidad en función del número de votos relevantes, ya que correlaciona con el algoritmo del voto promedio, por lo que en la práctica son equivalente, pero finalmente decidimos utilizar este último.

3.2.3 Recomendador factorización de matrices

Para la implementación de este recomendador utilizamos una librería conocida como RankSys[1]. Esta tiene varios algoritmos para la implementación y evaluación de sistemas de recomendación, entre los que se encuentra el algoritmo de factorización de matrices. Partiendo de la implementación de este algoritmo en la librería, fuimos adaptando el código para poder obtener recomendaciones y actualizar los datos del recomendador con las nuevas valoraciones hechas, ya que, al no ser de implementación propia, seguía un diseño diferente al que hasta ese momento había seguido.

3.3 Conjunto de datos

El conjunto de datos que hemos utilizado para ejecutar, evaluar y comparar los recomendadores propuestos es conocido como CM100k[2]. Dicho conjunto contiene votaciones de canciones por parte de usuarios reales de una plataforma de crowdsourcing.

En concreto, CM100k presenta 103.584 valoraciones de 1.053 usuarios distintos sobre 1.084 canciones. En este conjunto de datos, todos los usuarios e ítems tienen aproximadamente el mismo número de valoraciones, alrededor de 100 cada usuario y cada ítem, siendo unos datos bastante equilibrados. Este es el motivo por el que hemos seleccionado dicho conjunto para comparar el rendimiento de los distintos recomendadores, porque no presenta sesgos en la distribución del número de votos que puedan artificialmente afectar a las recomendaciones o favorecer a ciertos algoritmos frente a otros, como sí lo hacen otros conjuntos de datos públicos del área (Cremoensi et al. 2010).

Las canciones están puntuadas en una escala del 1 a 4, siendo 1 la puntuación más baja y 4 la más alta. De cara a ejecutar y evaluar los recomendadores, consideramos no relevantes las dos primeras opciones (1 y 2) y relevantes las dos mayores (3 y 4). Igualmente, si un ítem recomendado no se encuentra valorado en los datos, se considera no relevante. Por ello, para igualar la condición "no tener valoración" con "ser no relevante", reasignamos un valor 0 de relevancia a las opciones 1 y 2 (no relevantes), mientras que escalamos las relevantes a 1 y 2. Por tanto, las puntuaciones iniciales quedarían así:

1 → 0
2 → 0
3 → 1
4 → 2

[1] La librería está disponible públicamente en http://ranksys.org/

[2] El conjunto de datos está disponible públicamente en http://ir.ii.uam.es/cm100k

3.4 Experimentos offline

El objetivo de la evaluación offline es verificar el comportamiento de los algoritmos que se proponen en este trabajo, como paso previo a la evaluación online con usuarios reales. Es por tanto deseable que el experimento offline refleje lo más fielmente posible las condiciones en las que posteriormente se va a realizar la evaluación online. Por ello, simulamos un entorno donde los usuarios votan lo que los recomendadores iterativamente les sugieren, suministrando de esta forma nuevas valoraciones a los algoritmos, que las añaden a sus datos de entrada con el objetivo de mejorar las subsiguientes recomendaciones.

La idea inicial es partir de una división aleatoria de los votos del conjunto de datos en entrenamiento y test, utilizando la parte de entrenamiento para que los recomendadores tengan información con la que realizar las recomendaciones y la parte de test para comprobar lo acertadas que estás son.

Para reflejar una situación donde nuevos usuarios se van incorporando al sistema, en lugar de hacer la partición sobre los ratings globalmente, aplicamos una partición por usuarios. Partimos por tanto de una división del conjunto de datos en entrenamiento y test de las siguientes características: un 80% de los usuarios son seleccionados aleatoriamente y todas sus valoraciones asignadas a entrenamiento, mientras que las valoraciones del 20% restante de usuarios son asignadas a test. Esto se corresponde con un estado avanzado del sistema, es decir, el sistema ya tiene registrados una gran cantidad de usuarios con sus correspondientes valoraciones (simulados por los usuarios cuyas valoraciones han sido asignadas a entrenamiento) y debe realizar recomendaciones a usuarios de los que todavía no ha recibido información (simulados por los usuarios cuyas valoraciones se encuentran todas en test). Este escenario refleja además las condiciones en las que realizamos pruebas "online" con usuarios reales, que describiremos en el capítulo 4.

Para la realización de los experimentos offline, decidimos además restringir la recomendación de cada usuario a los ítems con una valoración en test de dicho usuario, para así simular con mayor precisión el posterior proceso online, donde todos los ítems recomendados presentarán una valoración por parte del usuario.

Partiendo de la división 80-20 inicial, a los usuarios con votos en test se les recomienda, en cada iteración, un ítem cuya valoración se encuentra en test. En este caso, consideramos como una iteración ofrecer una recomendación a todos los usuarios del conjunto de test. Al finalizar la iteración, la valoración de dichos ítems se incorpora al conjunto de entrenamiento, eliminándolos de test y actualizando los recomendadores. De esta forma, es como si los usuarios estuvieran valorando el ítem de igual forma que si se realizara de manera online, pues el sistema dispone de dicha valoración en el conjunto de test. Como cada usuario tiene alrededor de 100 valoraciones, el experimento termina después de 100 épocas en las que dichas valoraciones van pasando del conjunto de test al de entrenamiento al ser recomendados los ítems a los que hacen referencia.

3.4.1 Selección de parámetros

En todos aquellos recomendadores que necesitan algún tipo de parámetro, antes de compararlos con los demás, realizamos varias pruebas para elegir los que son óptimos de cara a los resultados finales y al experimento online con usuarios reales. Para ello, efectuamos varias ejecuciones variando y combinando sus valores. Escogemos los valores del recall del recomendador para la época 50, ya que consideramos la mitad de la ejecución como un buen punto para comparar los parámetros.

El recomendador de factorización de matrices cuenta con cuatro parámetros. Uno de ellos, el número de iteraciones mejora los resultados cuando lo incrementamos, pero aumenta considerablemente también el tiempo de ejecución, por lo que tomamos un valor estándar de 20. Para los parámetros k, α y λ variamos sus valores hasta encontrar la mejor ejecución. Probamos con valores cercanos a los que suelen dar mejores resultados según la literatura (Hu et al, 2008). Así, para k probamos con 5, 10 y 20, para α con 1, 10 y 100, mientras que para λ usamos 0.1, 1 y 10. Así el resultado en la época 50 es el que se observa en la Figura 5:

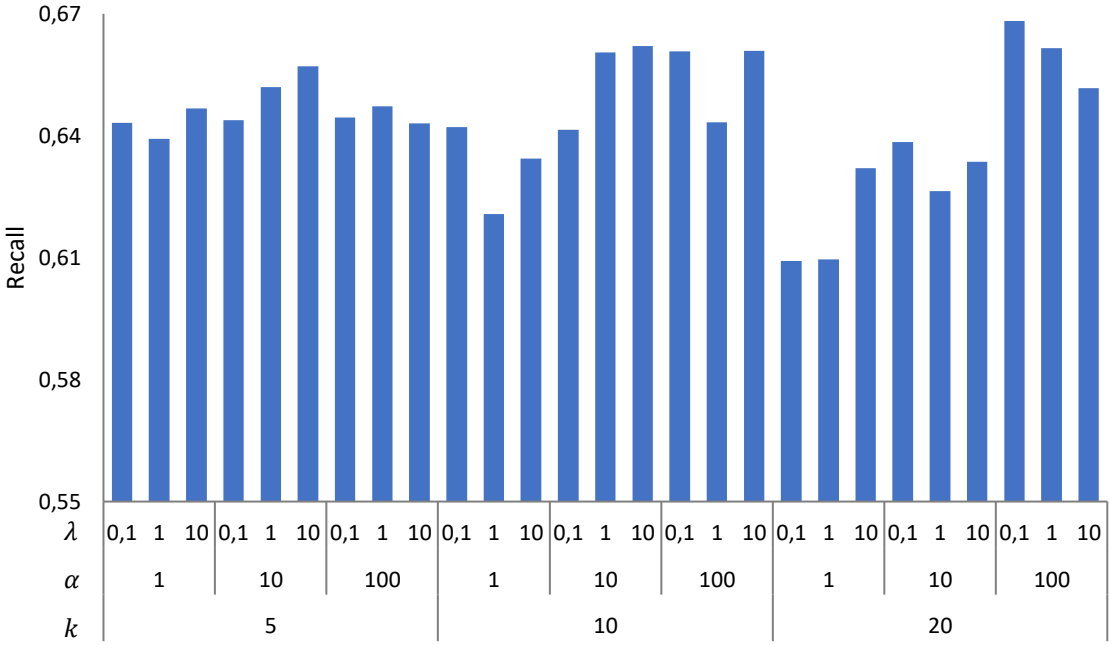

Figura 5: Barrido de parámetros recomendador factorización de matrices

Como se puede observar el punto con la mejor configuración es el que tiene los parámetros, ***k*=20, α=100, λ=0.1**, por lo que elegimos dichos valores para las ejecuciones del recomendador de factorización de matrices.

El recomendador de ϵ-greedy, únicamente tiene un parámetro, ϵ. Para decidir cuál utilizar, probamos a ejecutarlo varias veces variándolo entre 0.1 y 0.5, valores que se suelen utilizar en la literatura (Liang et al, 2014). Aunque se suelen usar también valores por debajo de 0.1, decidimos no elegir valores más bajos para evitar que el número de iteraciones en las que ϵ-greedy selecciona de forma aleatoria entre los brazos sea excesivamente bajo (un 0.1 de 100 iteraciones ya son únicamente 10 iteraciones) y no refleje por tanto la lógica que a grandes rasgos tiene el recomendador. En la figura 6, podemos observar el resultado en la época 50 al variar ϵ:

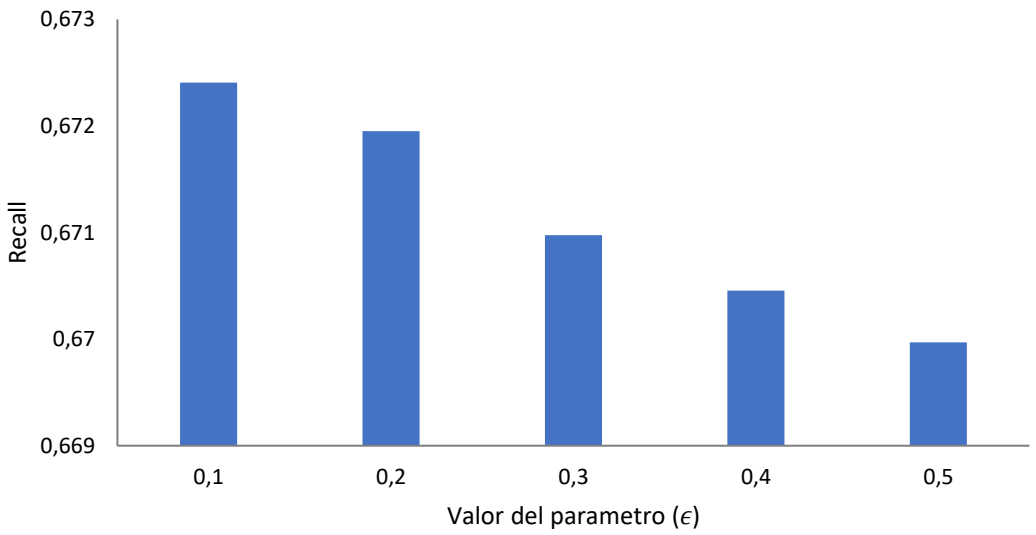

Figura 6: Barrido de parámetros recomendador ϵ-greedy

Se puede apreciar que ϵ = **0.1**, es el valor que mejores resultados obtiene, por lo que es el valor del parámetro elegido para este recomendador. Así en la mayoría de los casos (90%) realizará explotación, seleccionando el recomendador que mejor resultado está ofreciendo, mientras que en ciertas ocasiones (10%), realizará exploración, seleccionando aleatoriamente entre los candidatos, pudiendo encontrar algún ítem que maximice aún más la recompensa final.

Por último, el recomendador de Thompson sampling cuenta con dos parámetros, α y β. Los dos, son cruciales para el recomendador, ya que influyen tanto en la recompensa como en los cálculos posteriores para elegir el recomendador, por lo que es bastante importante elegir bien con que valores empezar. Partiendo de que los dos tienen que ser mayores que 0, probamos con los valores 1, 2, 5 y 10 para ambos parámetros, tal y como se ve reflejado en la Figura 7:

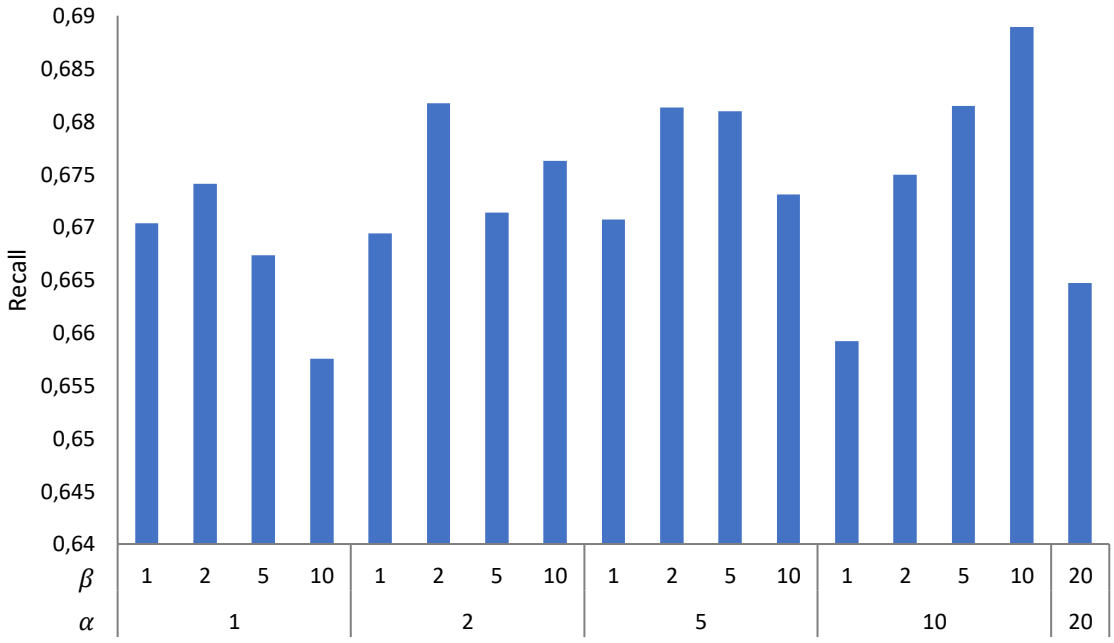

Figura 7: Barrido de parámetros recomendador Thompson sampling

Se puede observar cómo el mejor punto es el correspondiente con los parámetros, $\boldsymbol{\alpha}$**=10 y** $\boldsymbol{\beta}$**=10**, los cuales fueron por tanto escogidos para las pruebas. Para valores de α y β bajos, el recomendador tiende a elegir muchas veces el brazo con el que comienza, el cual coge mucho peso y no cambia, mientras que para valores altos suele ocurrir justo lo contrario, cambiando todo el rato de brazo. Por eso con valores intermedios y sobre todo cuando los dos parámetros coinciden, evitando probablemente que se produzcan sesgos, es dónde mejores resultados se alcanzan como se puede ver.

Como vemos en todas las gráficas expuestas, aunque hay diferencias que permiten decantarse por una u otra configuración de parámetros, todas las configuraciones son muy cercanas entre sí. Esto es debido a la restricción a test que realizamos, ya que, al estar limitándose siempre sólo a las valoraciones ya existentes en el conjunto de datos de test, reduce mucho las posibilidades en la recomendación, pero nos asegura elegir los adecuados de cara al experimento online.

3.4.2 Resultados

Estas pruebas tienen como objetivo evaluar el rendimiento de cada uno de los sistemas implementados de manera offline para validarlos y compararlos de cara al experimento online. Hay dos factores importantes que conviene observar. Uno es el número de ítems relevantes que cada recomendador es capaz de descubrir, mientras que el otro es la velocidad de descubrimiento con lo que lo hace, es decir, el número de iteraciones, en los que cada recomendador obtiene la información relevante que los usuarios en test desconocen.

El primero de ellos no es analizable mediante las pruebas offline, ya que, al restringir las recomendaciones al conjunto de datos de test, el número final de ítems relevantes recomendados por cada recomendador tras las 100 iteraciones será el mismo para todos los algoritmos, esto es, el número de ítems relevantes en test. En cambio, el segundo factor, la velocidad de descubrimiento, sí lo podemos analizar, ya que los ítems sí pueden ser descubiertos en un momento diferente según el recomendador utilizado. Este factor tiene mucha importancia, debido a que cuanto más lento se recomienden ítems relevantes, más probabilidades hay de que el usuario abandone el sistema porque no le gusten las recomendaciones.

3.4.2.1 Acierto acumulado

Para realizar el análisis, evaluamos utilizando la métrica recall. Utilizamos esta métrica de evaluación ya que es equivalente a una curva que muestre el número acumulado de aciertos, pero como conocemos el número total de ítems relevantes, podemos situarlo en el rango [0,1] en el cual se interpreta de una manera más clara. Así, la evolución del recall de cada recomendador en cada iteración (época) que va realizando el sistema, quedaría como vemos en la Figura 8:

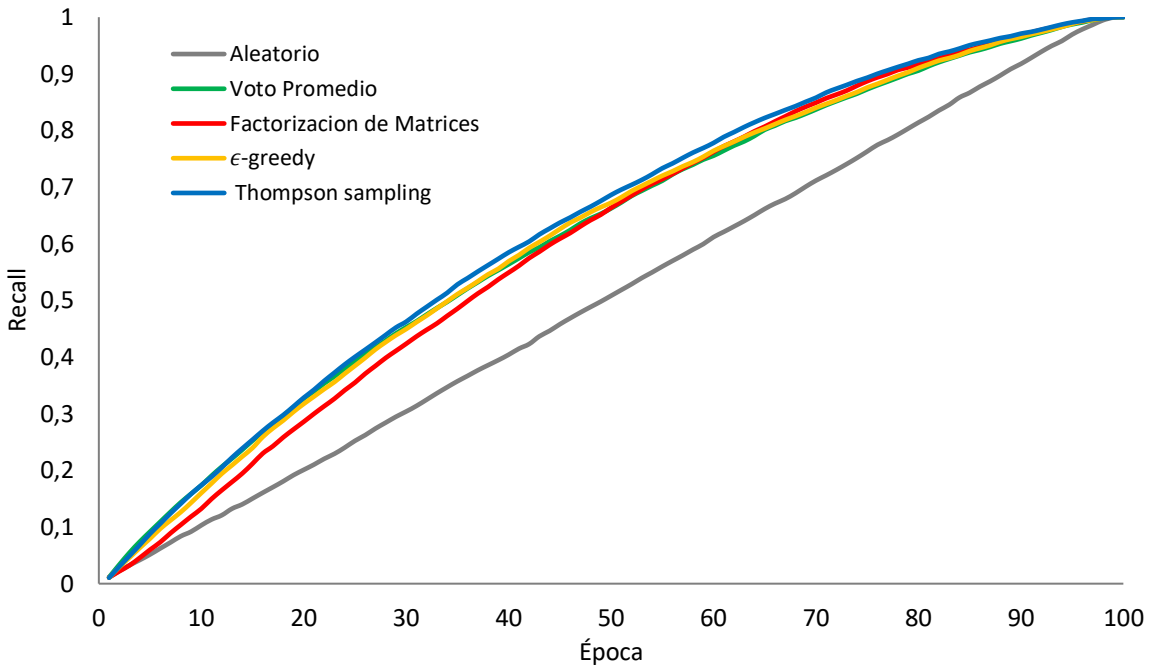

Figura 8: Evaluación offline de los recomendadores a lo largo del ciclo de recomendación iterativa.

Podemos ver como el aleatorio queda muy por debajo del resto de recomendadores, lo cual es esperable, ya que, al no seguir ninguna estrategia, tarda más en descubrir los ítems relevantes. Este nos puede servir de referencia para ver como mejoran cada uno de los recomendadores respecto a una recomendación base.

Entre los recomendadores restantes, sí que vemos que la diferencia existente entre ellos es mucho menor. Como ocurría con la elección de parámetros, esto es debido a la restricción a test, que deja un margen limitado a las recomendaciones. Por tanto, para analizar esa diferencia más en detalle, como hicimos en la selección de parámetros ya mencionada, nos situamos a mitad de la ejecución del experimento (época 50), donde ya es esperable que hayan podido aprender, para así poder observarlo más claramente:

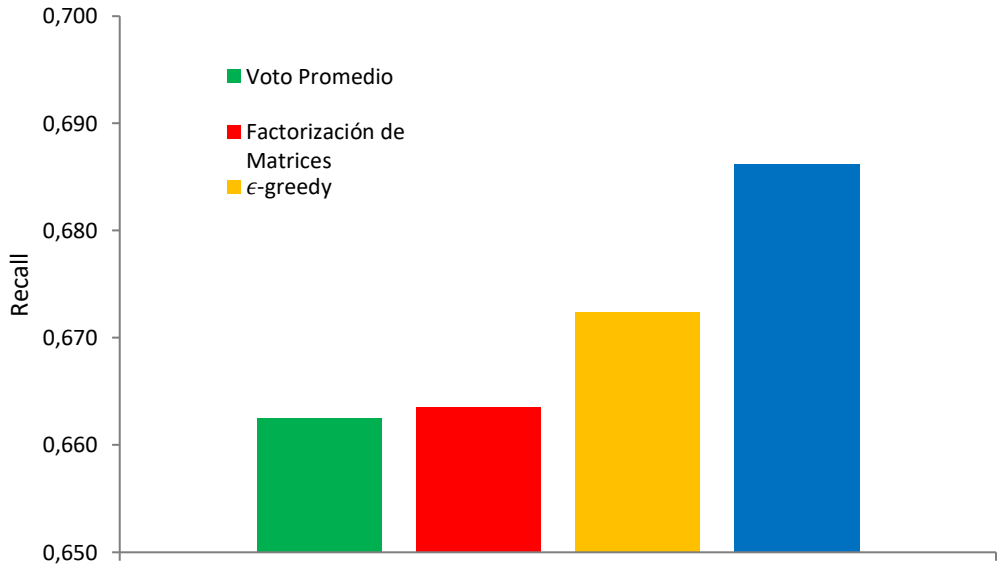

Figura 9: Acierto acumulado de los recomendadores en la época 50

Como se puede ver, hay dos recomendadores un poco más por debajo, muy igualados entre sí, los cuales son voto promedio y factorización de matrices. Al principio, el recomendador por voto promedio ofrece mejores resultados, ya que comienza ofreciendo los ítems que más le gustan a la gente en general, por lo que es esperable que muchos de ellos sean relevantes para una mayoría de usuarios. Pero según va avanzando el experimento, este se ve superado por el recomendador de factorización de matrices ya que, como personaliza según los gustos de cada usuario, cuanta más iteraciones pasen, más información posee sobre él, y puede producir mejores recomendaciones adaptadas a sus gustos personales.

Pero podemos observar que los recomendadores de ϵ-greedy y Thompson sampling quedan por encima de los descritos anteriormente. Esto es debido a que al combinar esos dos recomendadores, junto al uso que hacen de la recompensa de su propio algoritmo, consiguen solucionar en conjunto las carencias de cada uno de los recomendadores individuales que tienen como brazo, ya que en cada momento son capaces de detectar el algoritmo que está actuando mejor, y elegirlo. Detectan así la posibilidad de que distintos algoritmos sean el mejor en momentos distintos.

En el caso de la factorización de matrices las carencias aparecen en las primeras épocas, ya que no tiene información para poder personalizar sus recomendaciones. En voto promedio esto sucede al final, ya que se han ido eligiendo los ítems más populares para todos los usuarios y llega un momento en que las diferencias de popularidad entre los ítems restantes ya no son una señal lo suficientemente informativa acerca de su calidad.

3.4.2.2 Frecuencia de selección

Para contrastar este comportamiento, en las siguientes gráficas podemos ver el porcentaje de veces que (el porcentaje de usuarios para quienes) es elegido cada recomendador (voto promedio o factorización de matrices) en cada época y para cada uno de los recomendadores de aprendizaje por refuerzo (ϵ-greedy y Thompson sampling):

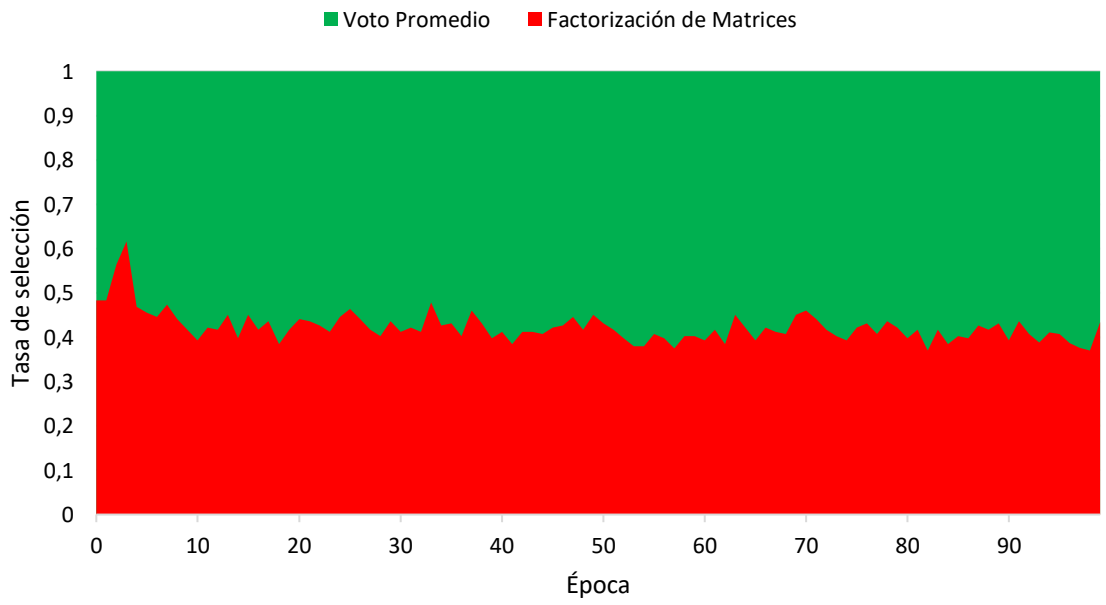

Figura 10: Promedio de elección de brazos de ϵ-greedy

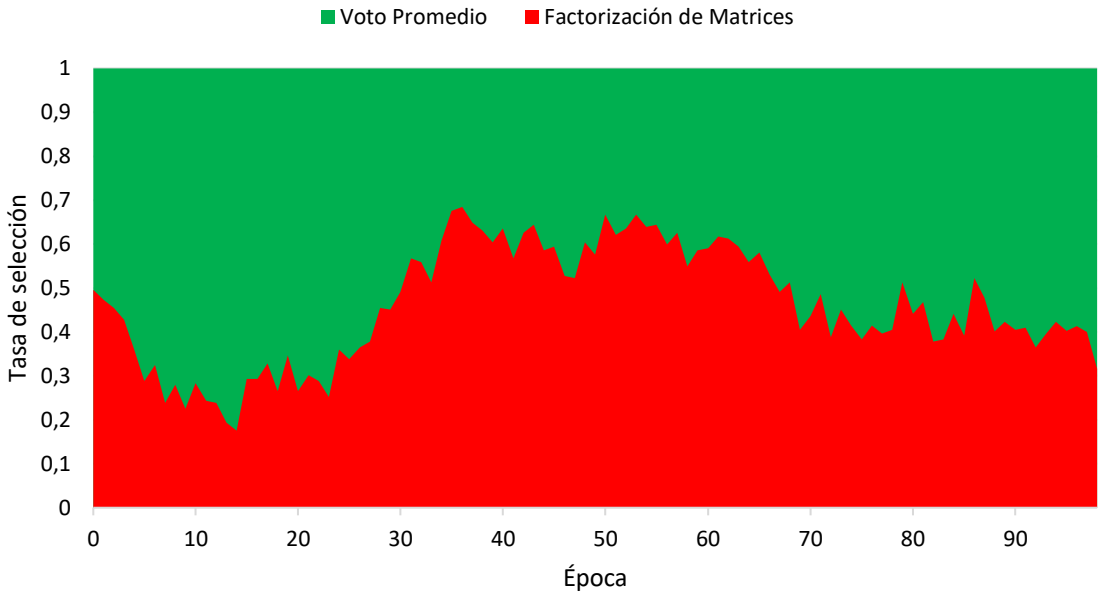

Figura 11: Promedio de elección de brazos de Thompson sampling

Como podemos observar en los dos casos, la selección de los brazos es más o menos equitativa, ya que se están combinando dos recomendadores con un rendimiento parecido. Como el algoritmo elige el brazo que mejores resultados esté produciendo en cada momento (en su mayoría de casos), nos asegura que se está mejorando el rendimiento para cada uno de ellos. El algoritmo de voto promedio ofrece al principio los ítems más populares para el conjunto de usuarios por lo que suele acertar casi siempre en su comienzo, mientras que la factorización de matrices a medida que obtiene más información del usuario ofrece mejores resultados. Podemos ver estos dos comportamientos reflejados en la figura 11 y aunque finalmente se acaba equilibrando parece que el recomendador de Thompson sampling está obteniendo mejores resultados, como hemos visto en la figura 9, debido a este comportamiento. La figura 10 toma una forma diferente al comportamiento mencionado, pero gracias a su algoritmo basado en el concepto de recompensa, también mejora el rendimiento de los recomendadores individuales que se analizan en la figura 9.

3.4.2.3 Comparación con combinación estática de algoritmos

Para asegurarnos que estos recomendadores están superando a los demás por aplicar aprendizaje por refuerzo y no porque sean combinaciones de algoritmos (ensembles), que ya es sabido que suelen mejorar a los algoritmos que combinan, los comparamos con un recomendador que hace un ensemble utilizando los mismos recomendadores, factorización de matrices y voto promedio. El elegido, cuyo funcionamiento ya introdujimos en el estado del arte, es Rank-sim. Este, como ya comentamos, combina los rankings de los dos recomendadores en uno único sumando la posición en la que se encuentra cada ítem en cada uno de los rankings.

Para poder observar mejor la comparación entre Rank-sim y los bandidos, medimos la diferencia de recall entre ellos para cada época, restando del recall de los recomendadores de aprendizaje por refuerzo el correspondiente valor de dicha métrica de Rank-sim, obteniendo el resultado que se muestra en las figuras 12 y 13. Nótese que estás gráficas muestran información equivalente al tipo de gráfica mostrada en la figura 8, pero centrándose en las diferencias, ampliándolas para apreciarlas mejor.

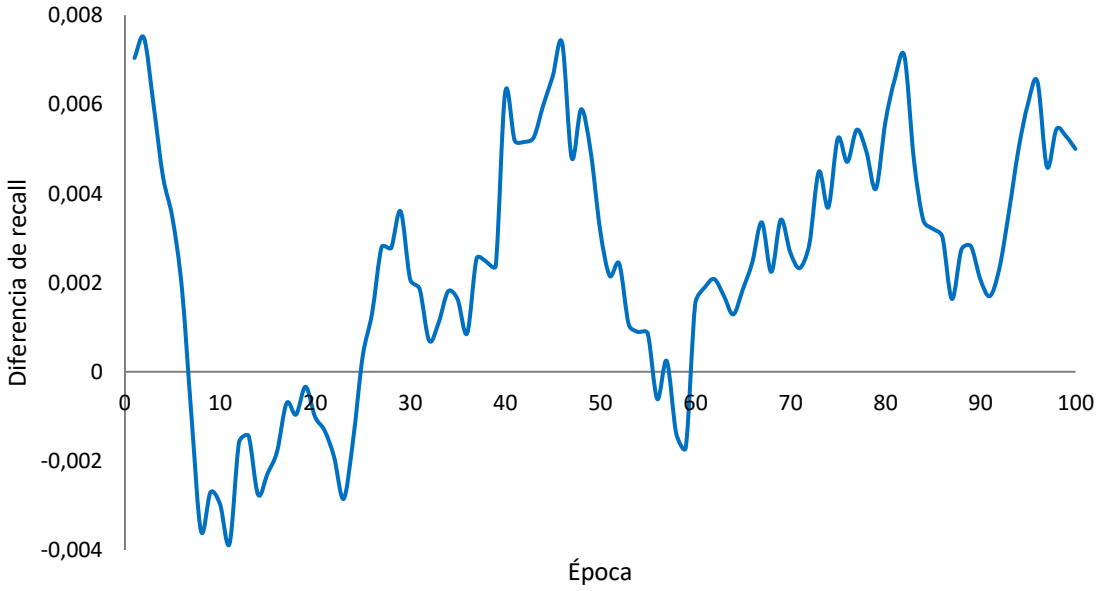

Figura 12: Diferencia de recall entre ϵ-greedy y Rank-sim

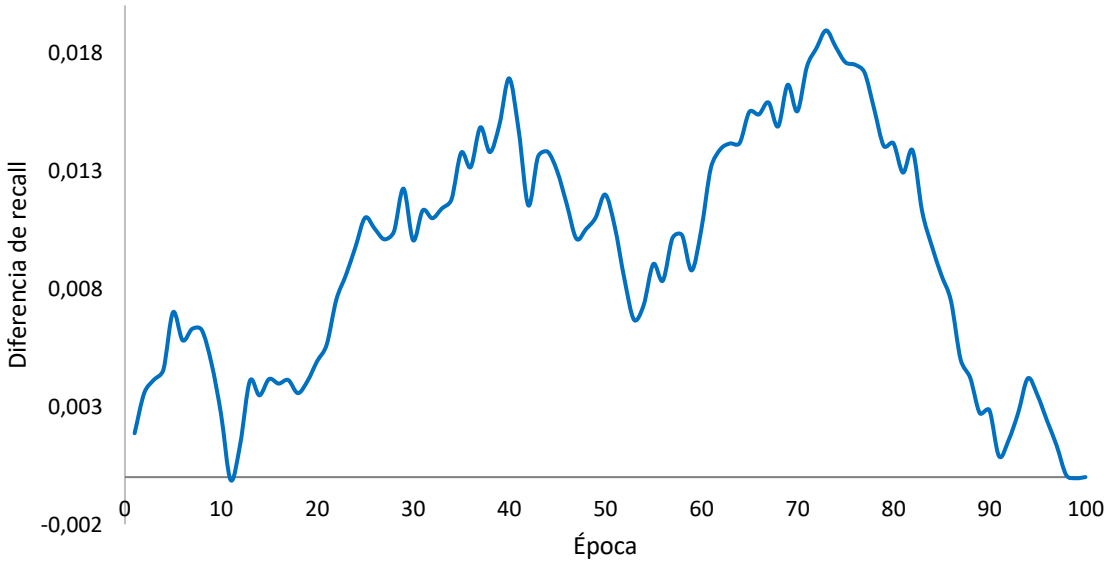

Figura 13: Diferencia de recall entre Thompson sampling y Rank-sim

Aunque en algún momento Rank-sim se acerca e incluso supera el rendimiento de los dos recomendadores (la curva desciende debajo de la diferencia cero), podemos ver en las gráficas como en general, tiene peor rendimiento (los valores de la diferencia son mayoritariamente positivos). Por tanto, esta nos viene a confirmar que efectivamente el empleo del aprendizaje por refuerzo y su concepto de recompensa que permite detectar el algoritmo que está funcionando mejor, sí que tiene una influencia positiva en el uso de ensembles para una futura recomendación con usuarios reales. Además, al tener que ejecutar únicamente el recomendador del brazo elegido, también ofrecen mejor rendimiento que los ensembles

tradicionales, ya que estos necesitan ejecutar todos los recomendadores que combinan para obtener la recomendación de cada uno y posteriormente combinarla.

3.4.2.4 Pruebas sin restricción a los datos de test

Decidimos probar qué ocurriría con los experimentos realizados si quitáramos la restricción que impone que los ítems recomendados deben presentar una valoración en el conjunto de test. Si un ítem recomendado no está valorado en test lo consideraremos no relevante, como ya indicamos anteriormente. Para todos los recomendadores, ejecutamos 100 épocas, rango que analizamos en las pruebas anteriores y con el que las podemos contrastar. En la siguiente gráfica podemos ver como se comportaron cada uno de los recomendadores:

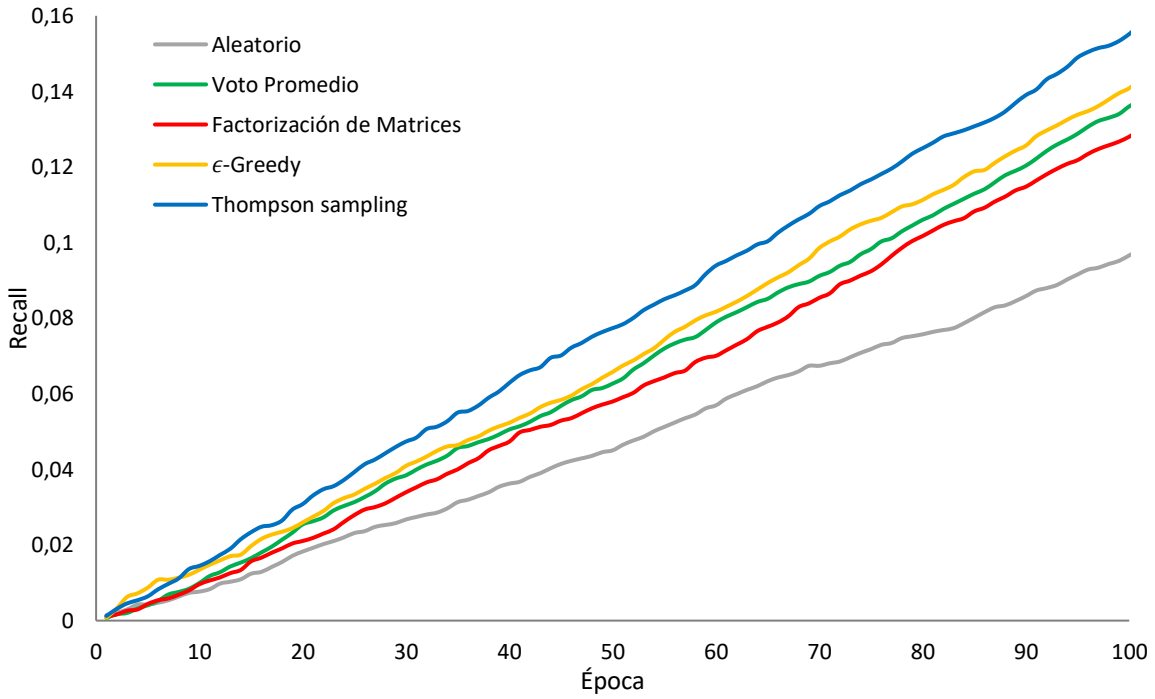

Figura 14: Evaluación offline del recall de cada uno de los recomendadores sin restricción a test

Como podemos ver, al igual que sucedía en las pruebas restringiendo a test, Thompson sampling es el recomendador que mejores resultados obtiene seguido de ϵ-greedy. Los dos superan a los recomendadores de referencia que utilizan como brazos, por lo que podemos confirmar la influencia positiva de los algoritmos basados en aprendizaje por refuerzo implementados en estos recomendadores. Además, al no restringir a test, las posibilidades de recomendación son más amplias y se ven ampliadas las diferencias entre cada uno de ellos.

En las figuras 15 y 16 podemos ver cómo se distribuye la selección de brazos por época, de forma análoga al caso de la restricción a test:

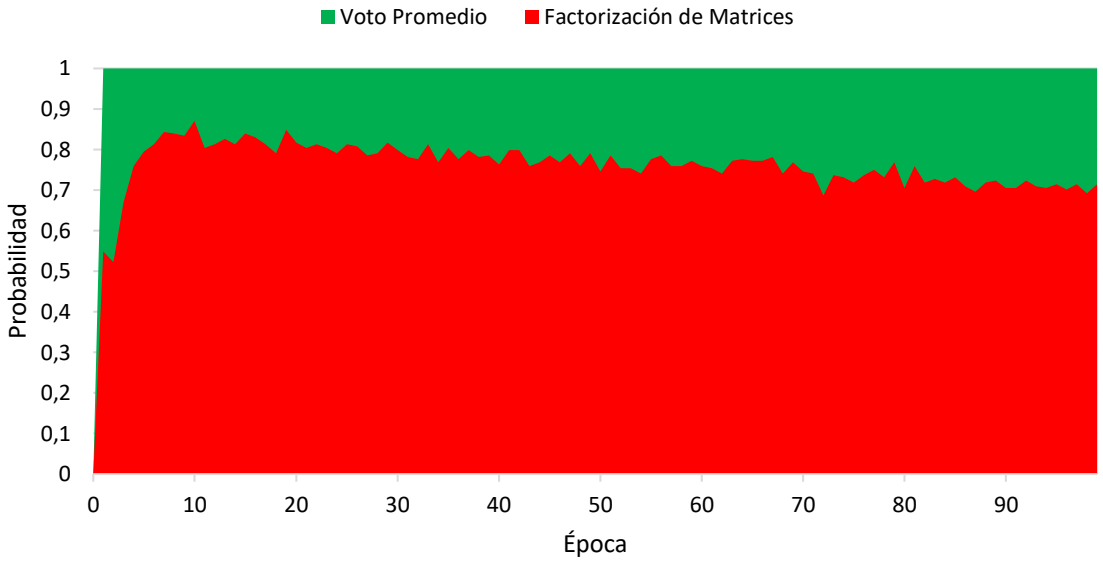

Figura 15: Promedio de elección de brazos de ϵ-greedy sin restricción a test

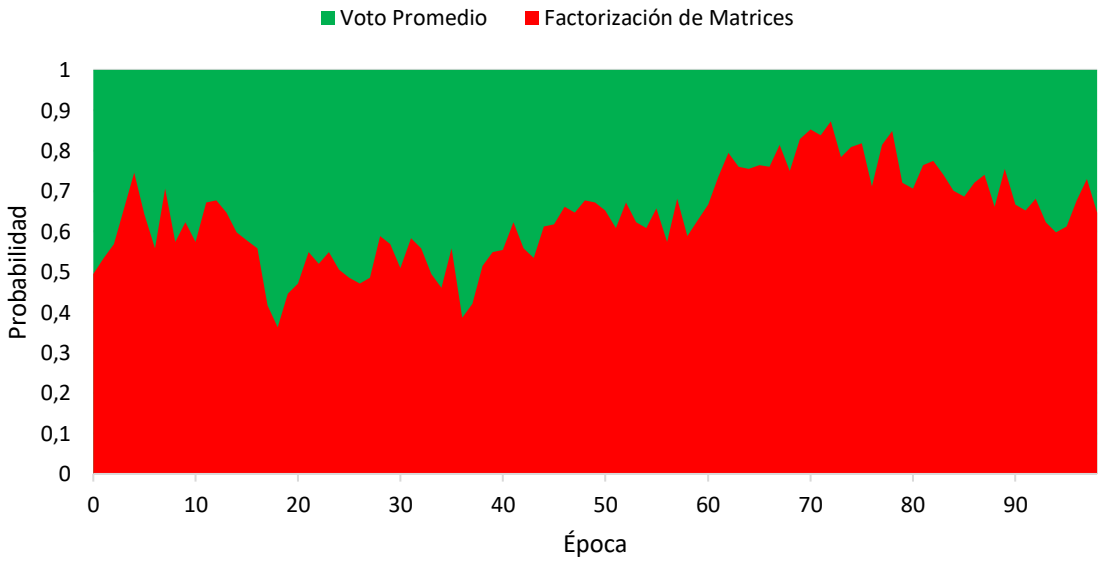

Figura 16: Promedio de elección de brazos de Thompson sampling sin restricción a test

En estas figuras, se aprecia más claramente como se pasa de recomendar con el voto promedio a recomendar con factorización de matrices, especialmente en la figura 15 de ϵ-greedy. Inicialmente parece elegirse más el recomendador de voto promedio, pero a medida que se suceden las épocas, el recomendador de factorización de matrices va adquiriendo mayor peso, ya que al personalizar conoce más información del usuario por lo que es esperable que acierte más. Esta adaptación dinámica para seleccionar el brazo óptimo permite solucionar las carencias que puedan tener en cada momento cualquiera de los brazos y, por tanto, alcanzar un resultado mejor que los otros recomendadores con los que estamos comparando.

En cuanto al comportamiento de los recomendadores de aprendizaje por refuerzo frente a Rank-sim, podemos observarlo en las figuras 17 y 18:

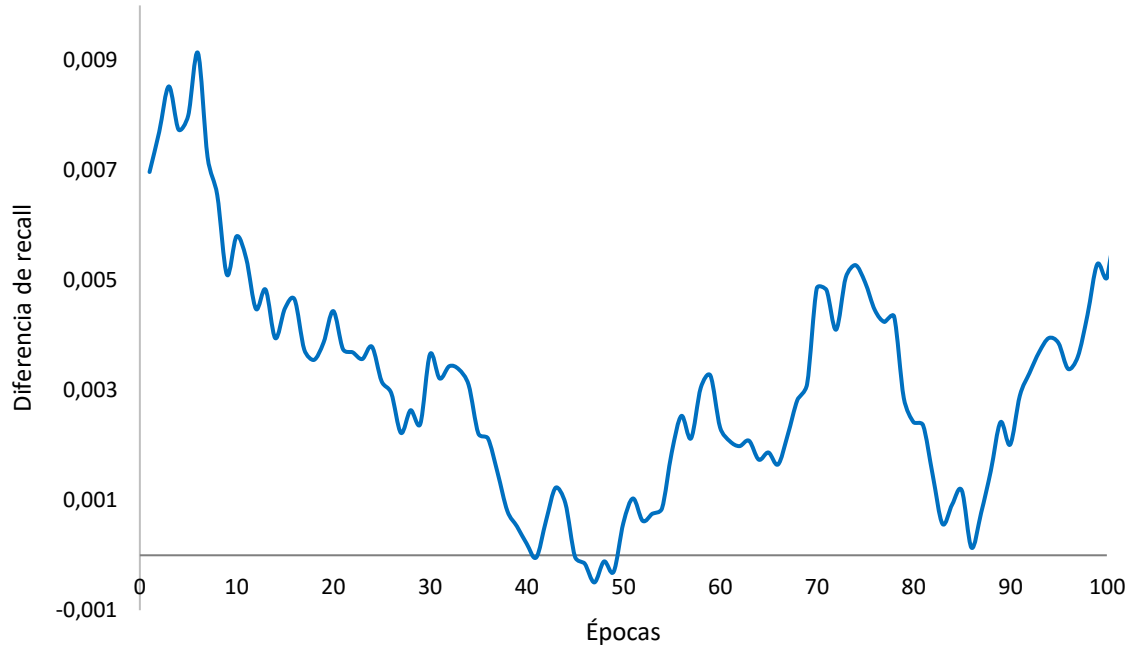

Figura 17: Diferencia de recall entre ϵ-greedy y rank-sim sin restricción a test

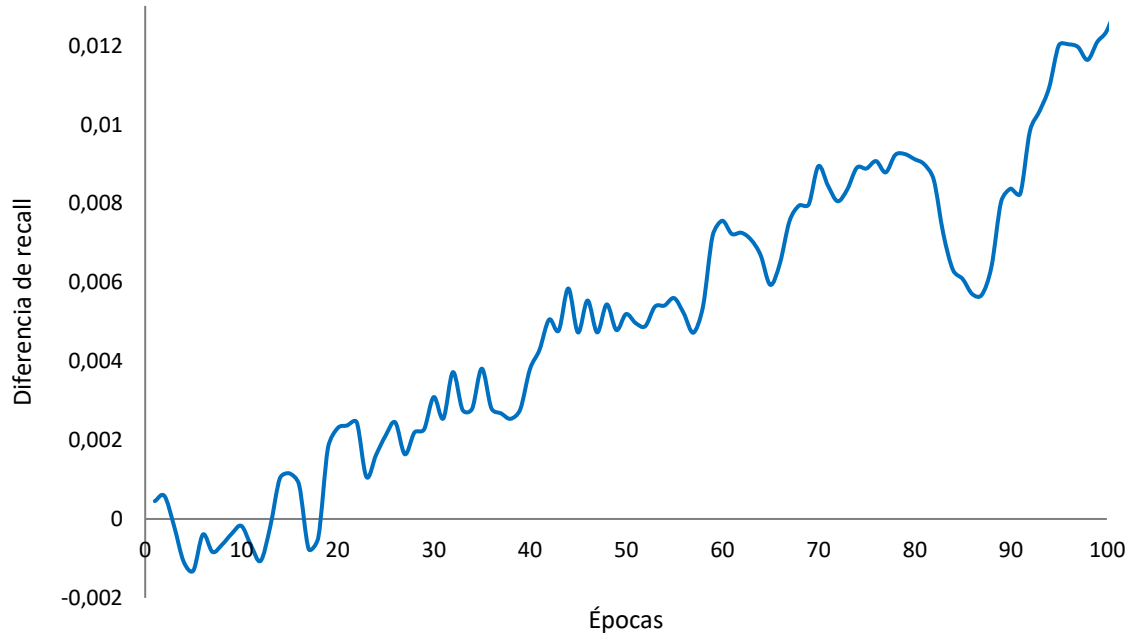

Figura 18: Diferencia de recall entre Thompson sampling y rank-sim sin restricción a test

Si nos fijamos en ϵ-greedy, podemos ver cómo al principio supera ampliamente el rendimiento de Rank-sim y según aumentan las épocas la diferencia se reduce, aunque sigue siendo superior. En cambio, en el caso de Thompson sampling, aunque comienza teniendo peores resultados, con el paso de las épocas la situación empieza a revertirse y cuanto más se avanza en el experimento mayor es la diferencia, lo que nos hace pensar que de continuar recomendando seguiría esta línea creciente. Aun con estas diferencias entre los dos bandidos, ya que al final son recomendadores distintos, ambos superan el rendimiento del

recomendador de Rank-sim, lo que nos viene a confirmar, como en las pruebas anteriores con recomendaciones restringidas a test, que ofrecen un mejor rendimiento.

Es cierto que todos estos experimentos sin restringir a test no son tan fiables de cara al experimento online como los que hemos hecho restringiendo, porque valorar todos los ítems que no están en test como no relevantes, aunque es el procedimiento estándar hoy en día para la evaluación offline, puede dar lugar a distorsiones, ya que es muy improbable que no haya algún ítem que le guste al usuario y sea considerado como relevante. Pero en este caso, sí nos sirven para confirmar el buen comportamiento de los recomendadores de aprendizaje por refuerzo que hemos analizado restringiendo a test, ya que podemos observar en un rango más amplio, como es la velocidad de descubrimiento de los ítems que sí son relevantes.

4 Aplicación

4.1 Plataforma

Para probar todos los recomendadores implementados de manera online, es decir, con usuarios reales, creamos una plataforma web que permite ofrecer y valorar sus recomendaciones. Esta, por tanto, deja a cualquier usuario escuchar y valorar las canciones del conjunto de datos CM100k, que le serán recomendadas una a una según el algoritmo de recomendación que le sea asignado al registrarse.

Para la creación de la plataforma, decidimos utilizar una arquitectura conocida como **Java EE** (Java Enterprise Edition) que nos permitía aprovechar el código desarrollado en Java de los experimentos offline. Esta plataforma consta de un conjunto de servicios, API y protocolos que proporcionan la funcionalidad necesaria para desarrollar aplicaciones web y para que estas puedan ser ejecutadas en un servidor de aplicaciones.

En esta arquitectura destacan dos elementos, los servlets y las páginas JSP. Los servlets se encargan de la lógica de la aplicación y permiten generar páginas web de forma dinámica a partir de los parámetros de la petición que es recibida por el servidor. Al recibir la petición un servlet puede, por ejemplo, procesar los datos de un formulario de la página que lo envía, realizar comprobaciones sobre esos datos, registrar una transacción o actualizar información de una base de datos entre otras cosas. Tras realizar las acciones correspondientes, devuelve la información al cliente generalmente a través de una página web implementada mediante un JSP. Las páginas JSP son por tanto como las páginas HTML, es decir, visualizaciones de la aplicación, pero pudiendo utilizar etiquetas especiales que nos permiten utilizar la sintaxis de Java.

Observando el comportamiento de estos elementos, se puede ver que esta arquitectura está basada en el patrón MVC (modelo-vista-controlador), el cual separa los datos y la lógica de negocio de una aplicación, de su representación y la gestión de sus eventos y comunicaciones. En este caso, las páginas JSP actúan como vista y los servlets como controlador. El modelo, sería una base de datos, la cual implementamos también para almacenar toda la información del conjunto de datos, así como de los recomendadores y usuarios registrados.

Para la gestión de la base de datos, así como para la conexión de la aplicación con ella, utilizamos MySQL, ya que es un sistema sencillo de utilizar, de código abierto y con soporte para java. En cuanto al diseño, repartí la información en cinco tablas:

- **Ratings**: Contiene toda la información sobre las valoraciones de los usuarios, tanto las que ya había (del conjunto de datos CM100k), como las que se van registrando al ir los usuarios realizando valoraciones. A parte de estas valoraciones, también tiene información sobre el recomendador que ha realizado la recomendación y la fecha y hora en la que se realiza la valoración. Además, en el caso de que la recomendación haya sido realizada por un algoritmo de tipo bandit, se indica también el algoritmo que ha sido seleccionado de entre los que lo conforman.
- **Tracks**: Almacena información sobre todas las canciones (ítems) que se pueden recomendar. Entre esta información está la url, para poder reproducirse la canción en la aplicación y el titulo o el artista.
- **Users**: Guarda la información de cada uno de los usuarios online registrados en la aplicación. Posee un identificador único para identificar cada usuario, además de su

nombre y apellidos, un usuario y una contraseña para entrar en la aplicación y un campo con el recomendador que le ha sido asignado. Es decir, al usuario se le asigna al registrarse un recomendador que será con el que se realizarán sus recomendaciones cada vez que haga uso de la plataforma. Utilizamos una estrategia Round-robin para ir asignando los recomendadores según se van registrando los usuarios en la aplicación. De esta manera nos aseguramos de que no se recomiende más veces con un recomendador que con otro, pues al final todos son sido asignados al mismo número de usuarios.

- **Alphabeta**: Para cada usuario que tiene asignado el recomendador Thompson sampling, esta tabla registra los valores de α y β para cada uno de los recomendadores de sus brazos, valores que se irán actualizando según el usuario vaya valorando las canciones que se le recomienden. Esos usuarios son identificados con el identificador asignado en la tabla users.
- **Egreedyreward**: Para cada usuario que utiliza el recomendador ϵ-greedy, esta tabla almacena y actualiza los aciertos y fallos producidos por cada uno de sus brazos para poder recalcular la recompensa en cada iteración. En este caso, los usuarios también son identificados con el identificador que tienen en la tabla users.

Para poder acceder a la aplicación vía web, tuvimos antes que desplegarla en un contenedor de aplicaciones. Para ello, utilizamos **Tomcat**, el cual se suele usar cuando trabajamos con una arquitectura JEE, ya que tiene soporte para los servlets y JSPs. Este contenedor a su vez se encontraba en un servidor remoto proporcionado por la autónoma en la dirección *ir.ii.uam.es:8181*. Una vez lo desplegamos en dicho contenedor, la aplicación pasó a estar accesible en *ir.ii.uam.es:8181/marcosTFG.*

Se puede entrar a la aplicación tanto desde el ordenador como desde el móvil, ya que configuramos la interfaz para que pudiera verse correctamente desde los dos dispositivos. Nada más abrirla, se nos muestra una pantalla para iniciar sesión en la aplicación, en la cual se solicita el usuario y contraseña para que cada persona pueda acceder a sus propias recomendaciones (Figura 19).

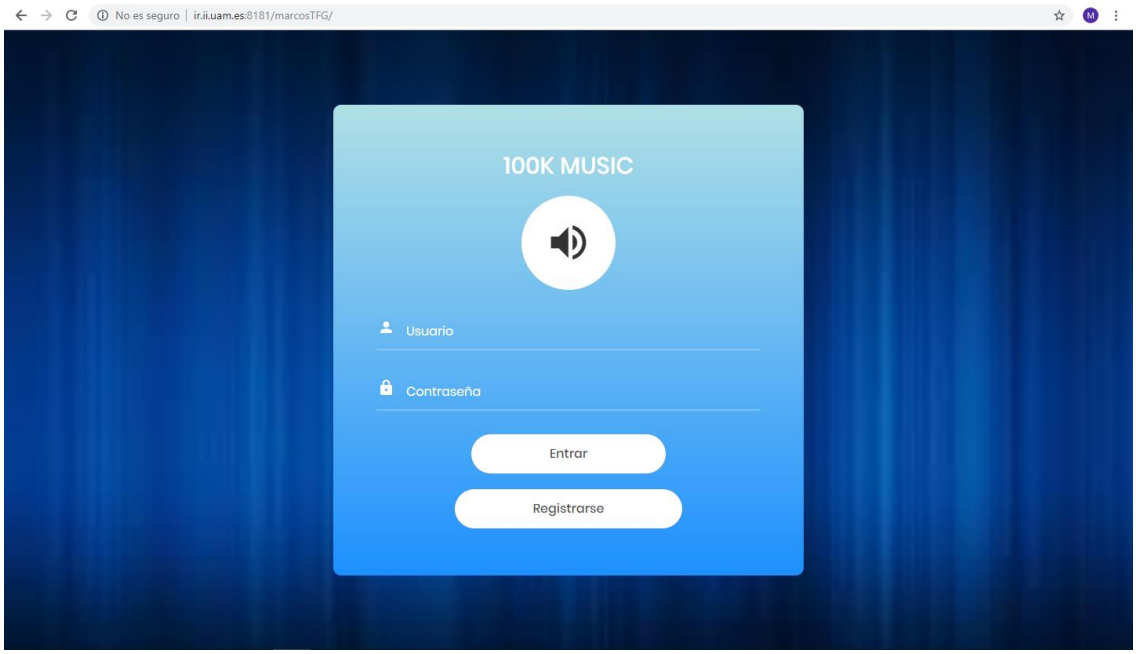

Figura 19: Pantalla principal (login) de la aplicación

En caso de que el usuario aún no se haya registrado en la aplicación, pinchando en el botón de “Registrarse” accede a una página para ello (Figura 20). En ella, puede introducir sus datos, los cuales serán usados en adelante para entrar en la aplicación. Una vez rellenados los campos y pulsado en “Entrar”, se accede directamente a la primera recomendación, sin tener que pasar por la página de inicio de sesión de nuevo.

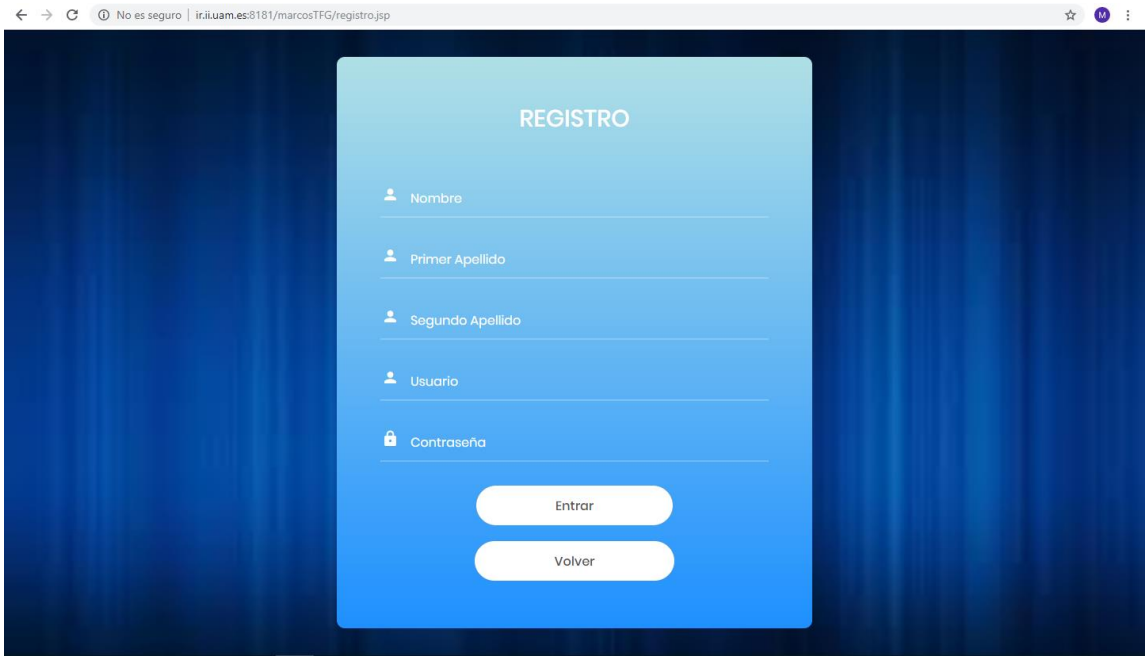

Figura 20: Pantalla de registro de la aplicación

Al haberse registrado o iniciado sesión correctamente, se redirecciona por tanto al usuario a la pantalla de recomendación, donde se le ofrece escuchar y valorar una canción (Figura 21). Además, también se le ofrece el número de canciones que lleva valoradas, ya que para el experimento solicitamos que fueran puntuadas al menos 100 canciones por usuario y de esta forma, ellos pueden llevar un control de las canciones que han valorado hasta el momento.

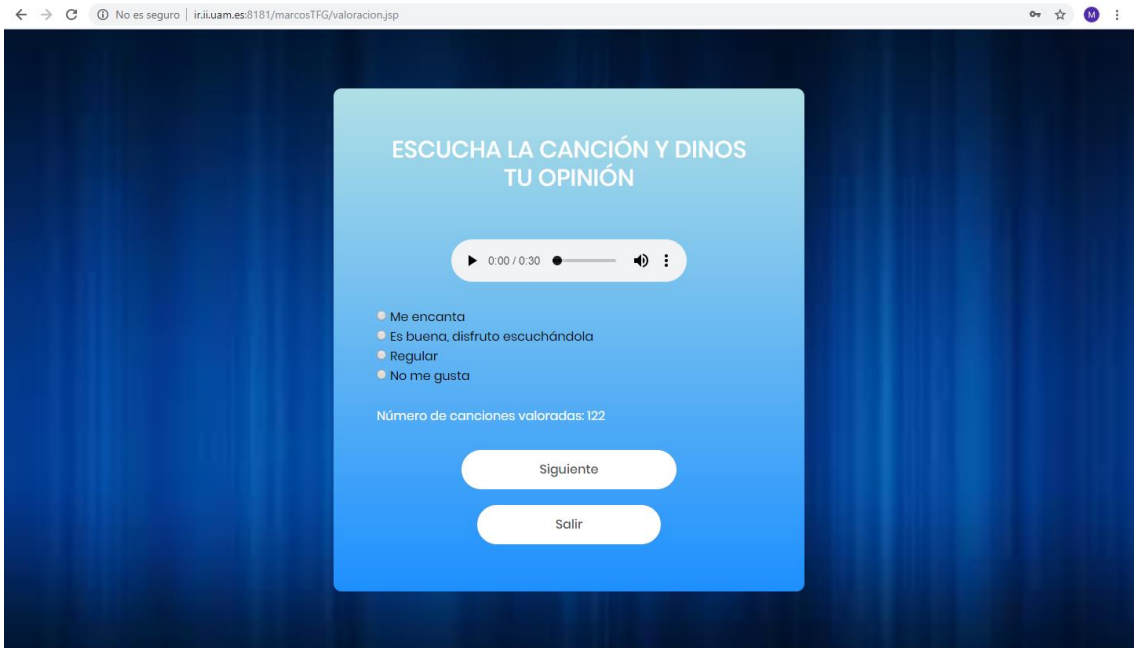

Figura 21: Pantalla de recomendación y valoración de la aplicación

Seleccionando la opinión que tiene sobre la canción y pulsando sobre “Siguiente”, el usuario puede ir valorando tantas canciones como quiera y en caso de no querer escuchar más, salir de la aplicación pulsando “Salir”, pudiendo volver a entrar cuando quiera accediendo a la dirección antes mencionada.

Cabe destacar, que en caso de que el usuario no reproduzca la canción, aunque se vuelva a recargar la página, no se tiene en cuenta esa valoración, volviéndola a recomendar hasta que el usuario sí la escuche. También, en el caso de que ninguna de las opciones sea seleccionada o alguno de los campos de las pantallas anteriores no se rellene, la aplicación se encargará de recordárselo al usuario. Además de estos detalles, también introdujimos algún otro control de errores esperable, como comprobar que el usuario y contraseña coincida con el guardado en base de datos, o que al registrarse no se permita un nombre de usuario si este ya había sido utilizado por otro.

4.2 Experimentos online

Para la realización de estos experimentos, obtuve los datos de usuarios reales, los cuales se registraron en mi aplicación y fueron recibiendo y valorando las recomendaciones de canciones que la aplicación les iba ofreciendo según el recomendador que les había sido asignado. Al haber implementado 5 recomendadores, decidimos que un número óptimo de usuarios por recomendador era 10, lo cual implicaba un total de 50 usuarios. Óptimo en el sentido de que es relativamente asequible el conseguir 50 usuarios y a su vez permite suficiente estadística para sacar conclusiones.

Solicitamos a cada uno de estos 50 usuarios que valorarán al menos 100 canciones, ya que consideramos que es un número lo suficientemente alto para obtener suficientemente información para los experimentos, pero que a su vez resulta asequible para los usuarios y evita que se cansen de realizar valoraciones y lo dejen. Además, este número también coincide con el número de ítems valorados por cada uno de los usuarios del conjunto de datos CM100k, por lo que permite que las pruebas se realicen en condiciones similares a las realizadas de manera offline, las cuales también se realizan sobre 100 valoraciones.

Sin embargo, las condiciones de las pruebas online difieren ligeramente de las offline, ya que en una iteración no podemos ofrecer a la vez una recomendación a todos los usuarios, debido a que los usuarios intervienen en momentos distintos, por tanto, tenemos que ir uno a uno.

Además, para obtener y graficar los resultados, no podemos utilizar el recall, ya que, no podemos saber cuántos relevantes hay en aquellas canciones que no hemos recomendado. En CM100k lo aproximamos por el número de relevantes en el conjunto de test, que es el mismo para todos los algoritmos y por tanto no beneficia a unos frente a otros. Pero aquí, lo que un usuario vota depende del algoritmo, si un algoritmo descubre muchos relevantes, y dividimos por dicho número, le estaríamos quitando mérito frente a otro que descubre pocos y por tanto es dividido por un número significativamente menor. Por tanto, decidimos evaluar según el número de ítems relevantes descubiertos hasta esa época, lo que nos permite así, analizar el acierto y la velocidad de descubrimiento por usuario.

Para calcular la evolución del número de ítems relevantes para cada recomendador promediamos dicho valor en cada una de las épocas y sobre los usuarios a los que dicho recomendador es asignado. Tras estos cálculos, podemos ver el resultado de compararlos en la figura 22:

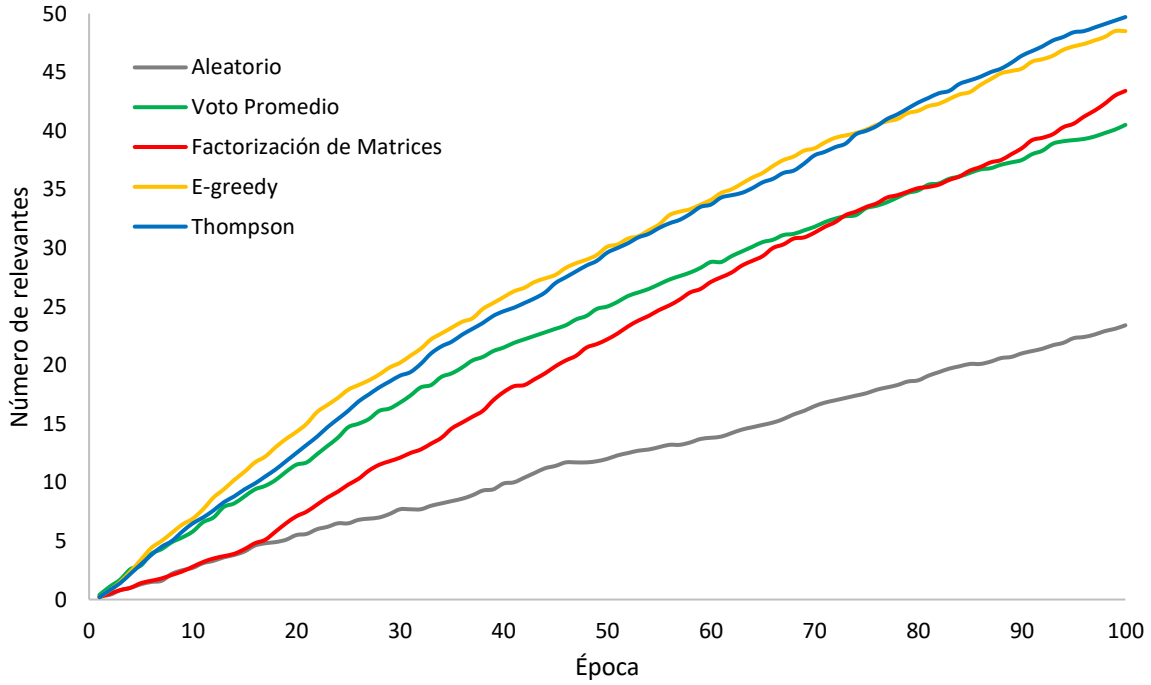

Figura 22: Evaluación online (plataforma) del número de aciertos de cada uno de los recomendadores

Podemos ver en la gráfica varias situaciones que se asemejan a los resultados de los experimentos offline. Lo primero es que el recomendador aleatorio comparado con todos los demás ofrece un pobre rendimiento, al no utilizar ninguna señal extraída de los datos para la recomendación.

Si observamos el recomendador de factorización de matrices, podemos ver que, al principio, al no tener información sobre los usuarios ofrece un rendimiento muy bajo, a la altura del aleatorio, pero conforme aumentan las épocas, va mejorando gracias a la personalización que aplica, superando más adelante al recomendador de voto promedio. En cambio, este último recomendador ofrece muy buenos resultados al comienzo, debido a que sabe identificar los ítems que en proporción más gustan al conjunto de usuarios, pero al ir recomendando dichos ítems los ítems restantes ya no gustan a una proporción tan amplia y el algoritmo pierde rendimiento.

Por último, podemos ver como los dos recomendadores que aplican aprendizaje por refuerzo, ϵ-greedy y Thompson sampling, están por encima de los recomendadores de referencia usados, obteniendo un mayor número de aciertos. Estos a diferencia del resto, siguen un camino muy uniforme, manteniendo más o menos un ritmo más alto y constante de descubrimiento. Esto es posible gracias al uso que hace del concepto de recompensa y del balanceo dinámico hacia el brazo con el recomendador óptimo en cada momento. Por tanto, vistos los resultados podemos afirmar que los recomendadores de aprendizaje por refuerzo implementados para las condiciones utilizadas, son mejores que los recomendadores de referencia que se suelen utilizar.

Aunque concuerdan con los del experimento offline realizado, en este caso podemos ver más claramente las diferencias que se producen entre todos los recomendadores evaluados, al no estar utilizando siempre las mismas valoraciones (cuando restringíamos a test), ni suponiendo ítems como no relevantes en caso de no ser conocida la opinión sobre ellos (cuando

no restringíamos a test). Además, al ser un experimento con usuarios reales, nos permite analizar el comportamiento de los recomendadores ante diferentes situaciones y usuarios, siendo un resultado más valioso de cara a un posible entorno de producción.

Como es de esperar, no todos los usuarios tendrán los mismos gustos por lo que a medida que van pasando las épocas y gracias a la capacidad que tienen los recomendadores de aprendizaje por refuerzo implementados, la selección de brazo para cada uno de los usuarios será diferente. Así, en las siguientes gráficas podemos ver como se distribuye dicha selección para los 10 usuarios que tiene cada uno de estos recomendadores:

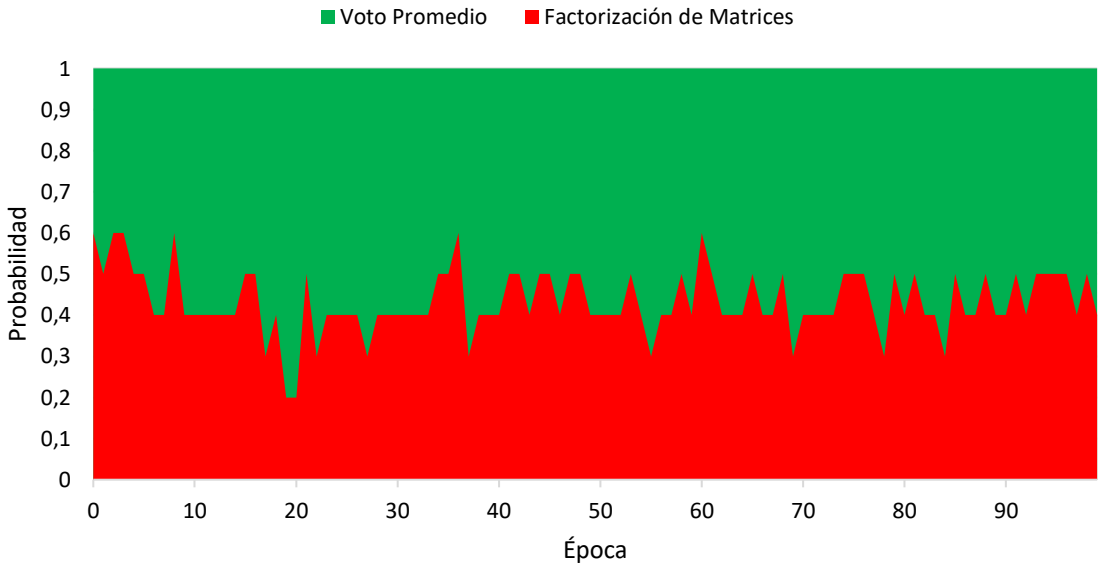

Figura 23: Promedio de elección de brazos de ϵ-greedy en el experimento online (plataforma)

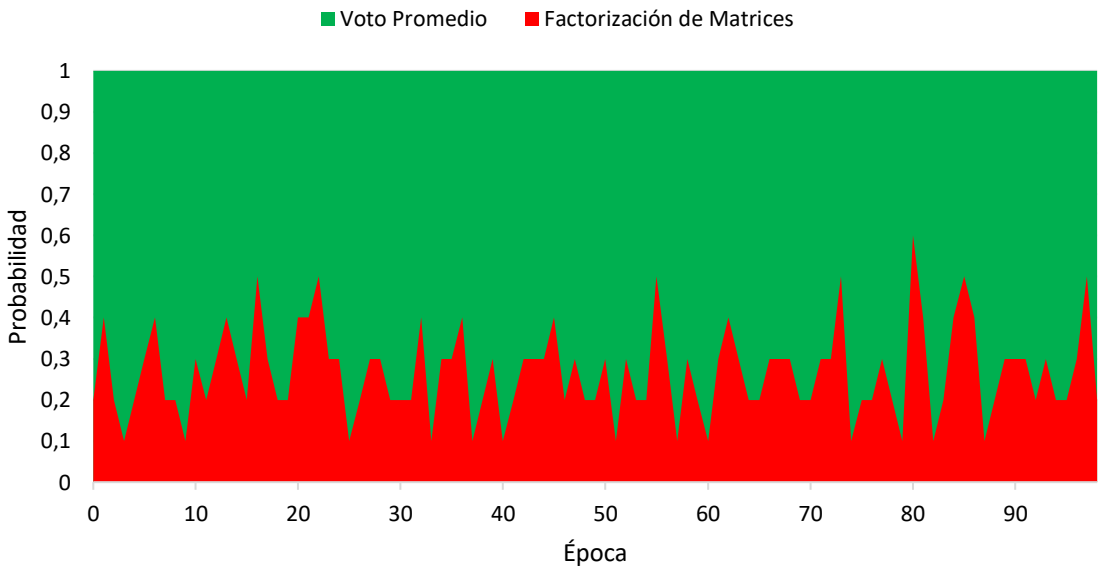

Figura 24: Promedio de elección de brazos de Thompson sampling en el experimento online (plataforma)

Como podemos observar, en ninguno de los dos casos la selección de brazo acaba tendiendo hacia un único recomendador, sino que siempre se distribuye en mayor o menor medida entre cada uno de ellos. Esta es una de las grandes ventajas que presenta este tipo de recomendadores frente a otro tipo de ejecuciones, como por ejemplo los test A/B ya que, a diferencia de estos, no acaban tendiendo a una de las posibles opciones hasta quedarse únicamente con ella, sino que van adaptando esta selección según las necesidades o debilidades que pueda tener alguno de sus brazos.

Por tanto, gracias a esto, consiguen sacar el mejor rendimiento posible al recomendador en cada momento, además de hacerlo de una manera eficiente dejando la selección del mejor candidato en manos del valor de la recompensa y no siendo necesario ejecutar todos los candidatos para combinar o comparar sus recomendaciones, lo cual es mucho más costoso y crece linealmente con el número de brazos.

Vemos que en el caso de ϵ-greedy, esta selección es más uniforme, estando más o menos igual de repartidos los brazos, tendiendo quizás un poco más al brazo de voto promedio. En cambio, aunque en algún caso se selecciona el brazo de factorización de matrices, vemos que Thompson sampling utiliza más el brazo de voto promedio. Esa mayor influencia del brazo de voto promedio en el recomendador de Thompson sampling, provoca que en este caso tenga peor rendimiento que ϵ-greedy como vemos en la figura 22, ya que acaba tendiendo más hacía el rendimiento del recomendador de voto promedio, aunque poco a poco debido a la influencia del brazo de factorización de matrices vaya mejorando.

A diferencia de los experimentos offline, no comparamos estos recomendadores de aprendizaje por refuerzo con Rank-sim u otro recomendador que aplique algún tipo de ensemble. Esto es debido a que este tipo de recomendadores tardan mucho en procesar la recomendación a devolver, al tener que generar un ranking para cada uno de los recomendadores que combina y posteriormente unir en un único ranking cada uno de ellos.

Ese tiempo de procesamiento para un entorno online, puede considerarse demasiado lento, lo que puede provocar que un usuario se canse de esperar a que cargue la página con la recomendación y, por tanto, deje la aplicación antes de terminar todas las valoraciones que solicitamos, por lo que decidimos no utilizarlo en la plataforma de recomendación.

Aunque no podemos tener por tanto esa comparación con otro tipo de recomendadores que combinen varios de ellos más simples, esto también nos demuestra la calidad de los recomendadores de aprendizaje por refuerzo implementados, ya que al solo necesitar ejecutar un único recomendador de los que utiliza en sus brazos, el tiempo que tarda en recomendar es igual que cualquiera de ellos, pero aprovechado todas las características de la combinación de los recomendadores.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

En este trabajo proponemos una forma de diseñar ensembles de recomendadores que emplea las técnicas de aprendizaje por refuerzo basadas en bandidos multi-brazo, y por tanto se beneficia de su capacidad para adaptarse dinámicamente al candidato que mejor esté funcionando. Dichos brazos están formados por distintos recomendadores básicos. El brazo elegido para la recomendación siempre es seleccionado dinámicamente basándose en el acierto obtenido en las recomendaciones anteriores.

Todos los experimentos realizados están orientados a comprobar la eficiencia de estos recomendadores de cara a una posible implantación de una plataforma de recomendación. Se realizaron primeramente pruebas offline para simular el comportamiento en un sistema real de recomendación, para posteriormente realizar las pruebas de manera online con la plataforma ya implementada. Estas pruebas online fueron realizadas por un total de 50 usuarios reales, cada uno de los cuales llevó a cabo 100 valoraciones. Estas son unas dimensiones pequeñas (especialmente el número de usuarios) pero suficientes para sacar conclusiones estadísticas que evidencien lo que pasaría a mayor escala

Una primera conclusión que se deriva de los experimentos anteriores es que los recomendadores basados en aprendizaje por refuerzo no sólo mejoran el comportamiento de los algoritmos individuales, sino que también superan la efectividad de otros recomendadores que los combinan.

Además de esta mejora en cuanto a resultados, estos recomendadores también presentan una ventaja computacional, ya que únicamente necesitan ejecutar un solo recomendador de los que componen sus brazos, mientras que otros recomendadores combinados necesitan procesar todos ellos para poder calcular su ranking combinado y ofrecer las correspondientes recomendaciones. Esta diferencia, sobre todo para conjuntos grandes y para ensembles de un número amplio de recomendadores, puede ser muy determinante, ahorrando una cantidad elevada tanto de tiempo como de procesamiento.

Los recomendadores de aprendizaje por refuerzo basados en bandidos multi-brazo cumplen por tanto el requisito de eficiencia, indispensable para ser considerados como alternativa de los test A/B. Además, en los casos como este en el que entre los recomendadores de los brazos no hay un claro ganador (como hemos podido ver en las gráficas de selección de brazos a lo largo del trabajo), utilizar este tipo de recomendadores puede mejorar los resultados de la selección de un único algoritmo basada en un test A/B. Estos últimos tradicionalmente se decantan, pasado un cierto tiempo, por una única de las opciones, ignorando la posibilidad de que en otro momento posterior sea otra alternativa la que funcione mejor (Siroker et Koomen, 2015).

Podemos concluir entonces, que este tipo de recomendadores que aplican aprendizaje por refuerzo, parecen mejorar tanto en resultados como rendimiento a otros que utilizan las técnicas de recomendación más comunes, por lo que sería interesante seguir investigando en esta línea, para ver si se obtienen todavía mejores resultados.

5.2 Trabajo futuro

Debido a la magnitud del trabajo, hay algunas cosas pendientes sobre las que se podría continuar trabajando para cerciorarnos de, e incluso mejorar, la eficacia de los recomendadores de aprendizaje por refuerzo con los que hemos estado trabajando.

Exponemos a continuación las posibles líneas a desarrollar en un futuro a partir del análisis de los recomendadores de aprendizaje por refuerzo llevado a cabo en de este trabajo:

- **Extender el análisis a otros conjuntos de datos y dominios** para verificar el buen rendimiento de los recomendadores de aprendizaje por refuerzo implementados. Esto es más factible para los experimentos offline, pero también sería aplicable en los experimentos online si se utilizan conjuntos o dominios en los que la opinión pueda formarse en cuestión de segundos o minutos como mucho.

- **Ampliar el número de brazos** para incluir otros recomendadores de referencia de filtrado colaborativo que conocemos, como por ejemplo kNN centrado en los usuarios o en los ítems. También, aunque esto implicaría mayores modificaciones tanto en la implementación como en los datos, podría probarse con recomendadores basados en contenido o incluso combinar estos con alguno de filtrado colaborativo.

- **Probar otras técnicas de recomendación** que utilizan aprendizaje por refuerzo. Por ejemplo, podríamos probar un recomendador basado en Softmax o UCB (Liang, 2017), para comprobar si alguno de ellos ofrece mejore resultado que los implementados.

- **Adaptar la plataforma** para que funcione como aplicación de recomendación al uso, del mismo modo que lo hace Spotify, así los usuarios podrían obtener información sobre el artista o el álbum de las canciones. De este modo, aunque nosotros lo estemos utilizando para realizar pruebas sobre los algoritmos de recomendación, podríamos hacerla más vistosa y así cumplir su fin que al final es satisfacer los gustos del usuario y permitirle descubrir nueva música que escuchar.

Referencias

G. Adomavicius and A. Tuzhilin. *Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions*, in IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, 734-749 (2005)

C.C. Aggarwal, C.C. *Recommender Systems: The Textbook*. 8-15. (2016)

B. Brodén, M. Hammar, B. Nilsson and D. Paraschakis. *Ensemble Recommendations via Thompson Sampling: An Experimental Study within e-Commerce*. 19-29 (2018)

R. Burke. *Hybrid Recommender Systems: Survey and Experiments*. User Modeling User-Adapted Interaction. 12:331-370 (2002).

R. Burke. *Hybrid Web Recommender Systems*. In: P. Brusilovsky, A. Kobsa and W. Nejdl. The Adaptive Web. Lecture Notes in Computer Science (2007).

R. Cañamares and P. Castells. *Characterization of Fair Experiments for Recommender System Evaluation – A Formal Analysis*. In Proceedings of the 12th ACM Conference on Recommender Systems (RecSys 2018) (2018)

P. Cremonesi, Y. Koren and R. Turrin. *Performance of recommender algorithms on top-n recommendation tasks*. In Proceedings of the fourth ACM conference on Recommender systems (RecSys '10). 39-46 (2010)

M. Elahi, F. Ricci and N. Rubens. *A survey of active learning in collaborative filtering recommender systems*. Computer Science Review (2016)

M. Elahi, F. Ricci and N. Rubens. *Active Learning in Collaborative Filtering Recommender Systems*. 113-124. (2014)

Y. Hu, Y. Koren and C. Volinsky. *Collaborative Filtering for Implicit Feedback Datasets*. Eighth IEEE International Conference on Data Mining, Pisa, 263-272 (2008)

H.L. Joon. *Analyses of multiple evidence combination*. In Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '97). 267-276 (1997)

D. Koulouriotis and A. Xanthopoulos. *Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems*. Applied Mathematics and Computation. 913-922 (2008)

J. Langford and T. Zhang. *The Epoch-Greedy Algorithm for Contextual Multi-armed Bandits*. Advances in Neural Information Processing Systems 20, Curran Associates, Inc. 817–824 (2008)

J. Leskovec, A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press. 308-309 (2014)

S. Li, A. Karatzoglou and C. Gentile (2016). *Collaborative Filtering Bandits*. In Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2016). 539-548 (2016)

Y. Liang. *An Ensemble Approach for News Recommendation Based on Contextual Bandit Algorithms* (2017)

D.M. Nichols. *Implicit Rating and Filtering* (1998)

P. Resnick and H.R.Varian. *Recommender Systems*. Communications of the ACM 40(3), 56–58 (1997)

A.I. Schein, A. Popescul, L.H. Ungar and D.M. Pennock. *Methods and metrics for cold-start recommendations*. In Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR ’02, 253–260 (2002)

D. Siroker and P. Koomen. *A/B testing: the most powerful way to turn clicks into customers*. John Wiley & Sons Inc (2015)

R. Sutton, S and A.G. Barto. *Introduction to Reinforcement Learning*. MIT Press (1998)

L. Tang, Y. Jiang, L. Li, and T. Li. *Ensemble contextual bandits for personalized recommendatio*n. In Proceedings of the 8th ACM Conference on Recommender systems (RecSys '14). 73-80 (2014)

D. Valcarce, A. Bellogín, J. Parapar and P. Castells. *On the robustness and discriminative power of information retrieval metrics for top-N recommendation*. In Proceedings of the 12th ACM Conference on Recommender Systems (RecSys 2018). 260-268 (2018)

Anexos

A Diagramas de clases

A continuación, mostramos el diagrama de clases del proyecto con las clases implementadas para la recomendación. Omitimos mostrar las clases que implementan la vista, el modelo y el controlador, ya que durante el trabajo se explica cómo se relacionan siendo así más clara la visualización del resto de clases que no han sido explicadas. Para que se vea mejor, realizamos una división en tres subsistemas: *ratings, ranking y recommenders*. Aunque los mostramos separados estos a su vez también están relacionados como a continuación explicaremos. Un *ranking* contiene un conjunto de *ratings* ordenados de mayor a menor mientras que un *recommender*, contiene un *ranking* para saber que ítem recomendar. En cuanto a las propias relaciones de cada subsistema, las podemos observar a continuación:

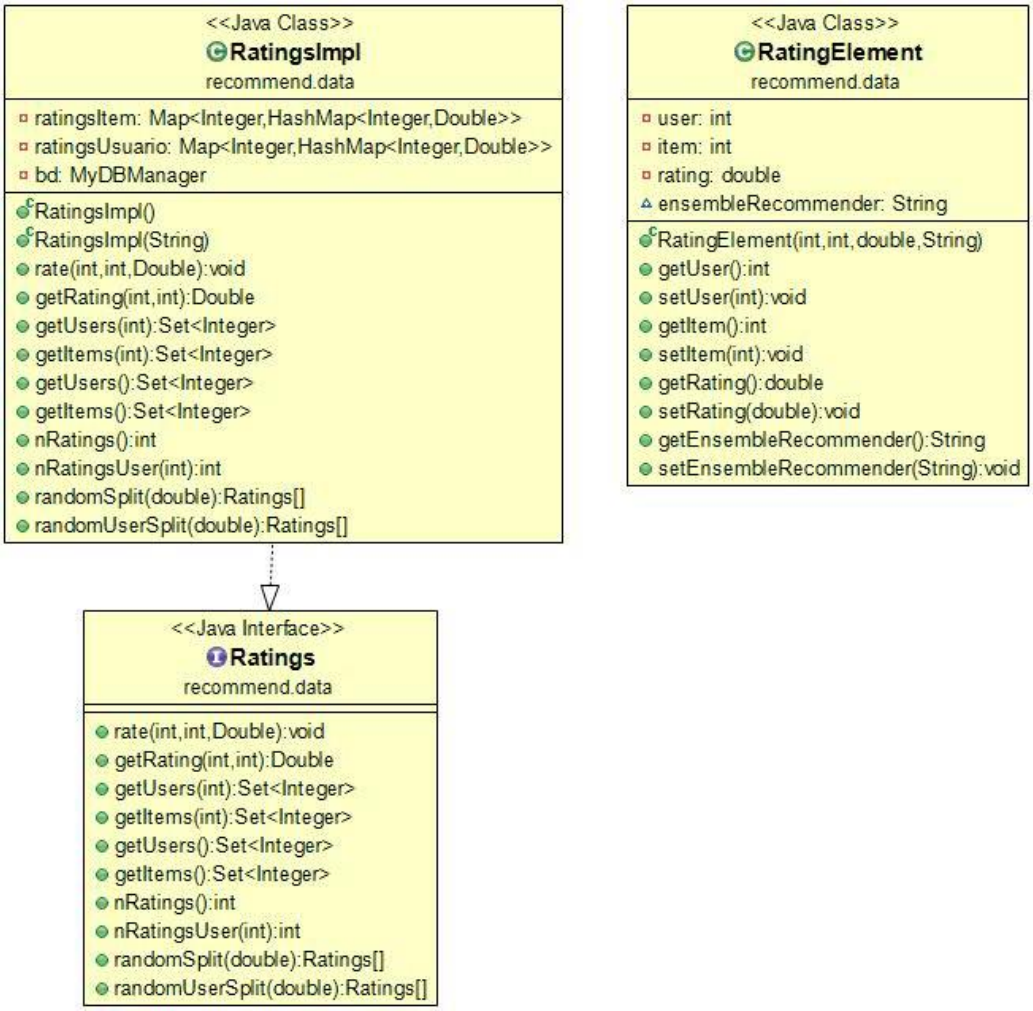

Figura 25: Diagrama de clases del subsistema de ratings

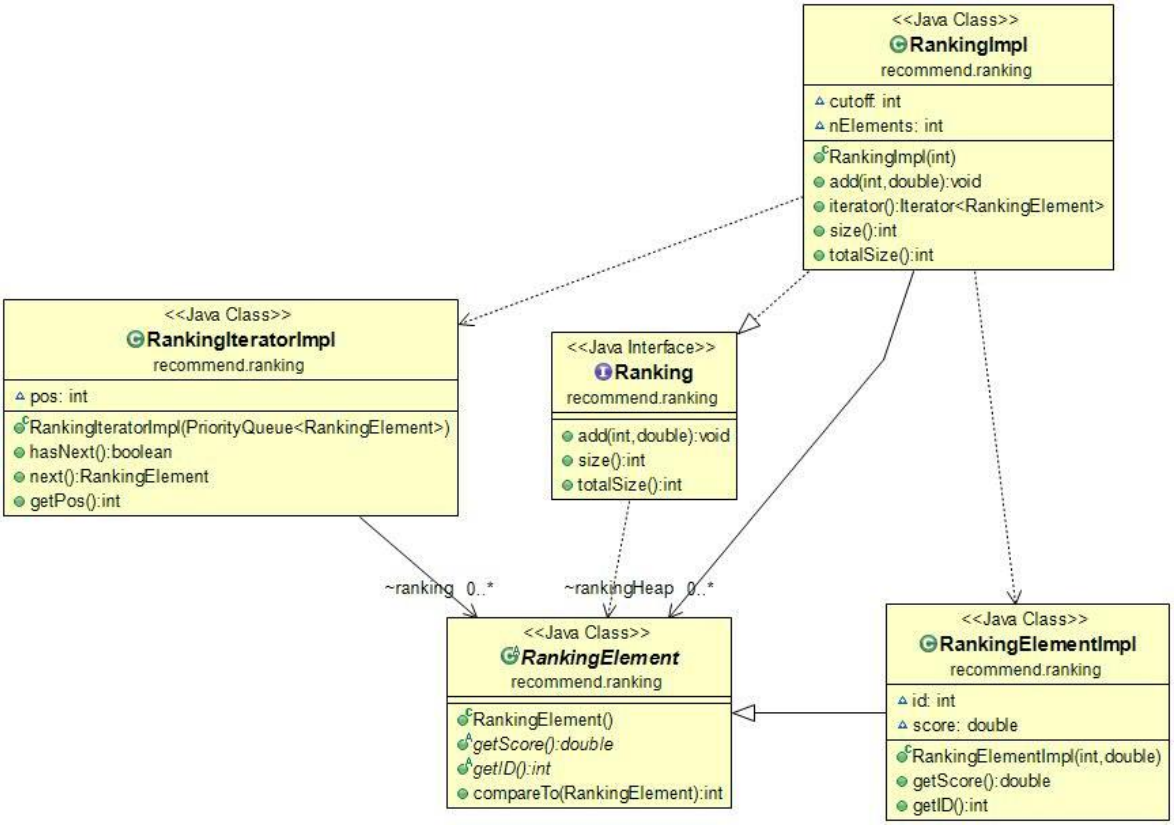

Figura 26: Diagrama de clases del subsistema de ranking

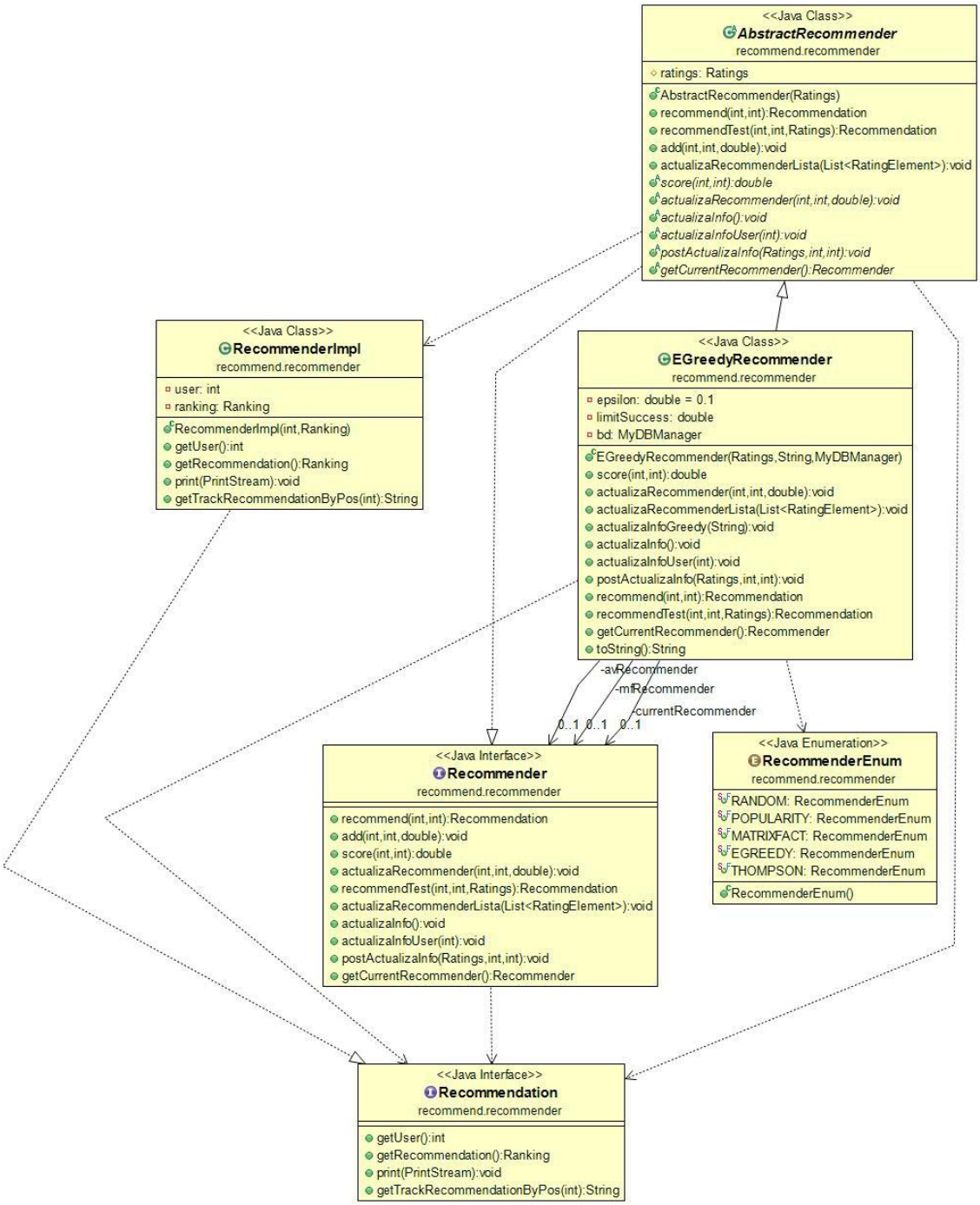

Figura 27: Diagrama de clases del subsistema de ranking

En este último subsistema cabe destacar que de los recomendadores implementados solo mostramos la clase de ϵ-greedy, para que sea una visualización más clara, pero los otros recomendadores implementados tendrían las mismas relaciones que este.